



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

„Entwicklung eines automatisierten Prozesses zur Verarbeitung von UAV-Bilddaten mittels Hough Circle Transform zur Erkennung von GCPs“

vorgelegt von

Christian Martens

u105638, UNIGIS MSc Jahrgang 2020

Betreuer:

Prof. Dr. Josef Strobl

Zur Erlangung des Grades
„Master of Science – MSc“

Wernigerode, 14.03.2023

Danksagung

Herzlich bedanken möchte ich mich beim gesamten UNIGIS-Team für die durchweg professionelle Betreuung während der letzten drei Jahre. Im Besonderen einen großen Dank an Regina und Eva, die auch auf die nicht so schlauen Fragen stets eine schlaue Antwort parat hatten. Ein weiterer Dank geht an Dr. Christian Neuwirth, der mir stets mit Rat und Tat zur Seite stand sowie natürlich an Prof. Dr. Josef Strobl für wertvolle Denkanstöße während der Betreuung dieser Arbeit.

Ein besonderer Dank geht an meinen Freund, Kollegen und Vorgesetzten Thomas Gudera, der mich auf diesem Weg stets motiviert und inspiriert hat und mit dem ich gemeinsam sämtliche Türen und Tore von der Idee dieses Studiums bis hin zur Ermöglichung des Themas für diese Arbeit öffnen konnte.

Ein spezieller Dank auch an meinen Freund und Kollegen Nico Laser, der mir unter anderem auch bei widrigstem Wetter gutgelaunt dabei half, weiße Farbkleckse auf den Boden zu sprühen. Für eine Liste aller Gefälligkeiten ist hier leider kein Platz.

Meinem Freund André Zieris an dieser Stelle meinen herzlichen Dank für die kreative Inspiration.

Auch an meinen ehemaligen Vorgesetzten Herrn Jan Kitschun sowie an meinen aktuellen Arbeitgeber, die Tegel Projekt GmbH, ein großes Dankeschön für die Unterstützung und Ermöglichung dieses Studiums.

Der größte Dank gebührt natürlich meiner Familie, allen voran meiner Partnerin Maren, die mich für diese Sache ermutigt und mir stets den Rücken freigehalten hat. Gleiches gilt für Angela, meiner Schwiegermutter in spe, vor allem für das Hüten unseres Sohnes Otto als die Zeit so langsam knapp wurde. Bei letzterem möchte ich mich abschließend auch bedanken, obwohl er noch viel zu klein ist und von der ganzen Angelegenheit wahrscheinlich nicht viel mitbekommen hat.

Zusammenfassung

Im Rahmen dieser Masterthesis wird ein automatisiertes Verfahren zur photogrammetrischen Auswertung und Weiterverarbeitung von UAV-Bilddaten zu einem präzisen Orthomosaik vorgestellt. Ein besonderer Schwerpunkt liegt dabei auf der genauigkeitssteigernden, automatischen Erkennung von kreisförmigen Ground Control Points (GCPs) durch die Hough Circle Transform Methode (HCT).

Es konnte eindeutig nachgewiesen werden, dass HCT mit einer Erkennungsrate von über 90 % eine robuste Methode zur Erkennung von Ground Control Points ist, wobei durch die hinzugewonnenen Erkenntnisse auch eine Erkennungsrate gegen 99% theoretisch möglich wäre.

Darüber wurde bewiesen, dass die Positionsgenauigkeit von photogrammetrisch erzeugten Daten mithilfe der implementierten HCT-Methode gegenüber einer Auswertung von reinen GNSS-RTK Daten ohne GCPs signifikant verbessert werden konnte.

Mithilfe von zwei Quadrocoptern wurden vier Gesamtbefliegungen eines ca. 450ha großen Areals an unterschiedlichen Tagen mittels Streifenbefliegung durchgeführt. Die gewonnenen Daten wurden durch das hier vorgestellte Verfahren jeweils vollautomatisch zu einem Orthomosaik mit einer Bodenauflösung (GSD) von 1,37cm prozessiert, sowie potenziellen Nutzern als Geodatendienst in einem WebGIS zugänglich gemacht. Insofern keine extremen Bedingungen wie Schneefall vorlagen, konnte eine Lagegenauigkeit von 1,5cm bis 2,5cm und eine Höhengenaugkeit von knapp 5cm erreicht werden.

Abstract

In this master thesis, an automated method for photogrammetric analysis and processing of UAV image data into a precise orthomosaic (DOP) is presented. A special focus is given to the accuracy-enhancing automatic detection of circular Ground Control Points (GCPs) by the Hough Circle Transform method (HCT).

It has been clearly demonstrated that HCT is a robust method for the detection of ground control points with a detection rate of more than 90%, and with the acquired knowledge a detection rate towards 99% is theoretically possible.

Furthermore, it was proven that the position accuracy of photogrammetrically generated data could be significantly improved using the implemented HCT method compared to an evaluation of pure GNSS RTK data without GCPs.

With the help of two quadcopters, four total flights of an area of about 450 ha were executed on different days by strip flights. The acquired data were processed fully automatically to an orthomosaic with a ground resolution (GSD) of 1.37cm and made available to potential users as a geodata service in a WebGIS. If no extreme conditions such as snowfall were present, a positional accuracy of 1.5cm to 2.5cm and a height accuracy of almost 5cm could be achieved.

Inhaltsverzeichnis

	<u>Seite</u>
Danksagung	I
Zusammenfassung	II
Abstract.....	III
Inhaltsverzeichnis	IV
Abkürzungsverzeichnis.....	VII
Abbildungsverzeichnis.....	VIII
Tabellenverzeichnis.....	XII
1 Einleitung	1
1.1 Der ehemalige Flughafen Berlin Tegel	1
1.2 Monitoring und Prozessunterstützung durch Drohnen.....	3
1.3 Stand der Forschung	9
1.4 Forschungsfrage	11
2 Methodik.....	12
2.1 Grundlagen	12
2.1.1 GNSS (Satellitennavigation)	12
2.1.1.1 RTK (Real Time Kinematic)	14
2.1.2 Photogrammetrie (Structure from Motion).....	17
2.1.3 Hough Circle Transform	19
2.2 Eingesetzte Ausrüstung	23
2.2.1 UAV-Plattform (DJI M300 RTK)	23
2.2.1.1 Photogrammetrie-Kamera (DJI Zenmuse P1).....	25
2.2.1.2 Fallschirm (AVSS PRS-300).....	26
2.2.2 GNSS-Messgerät (Leica Zeno FLX 100).....	26
2.2.3 Prozessierungsrechner.....	27
2.2.4 Software.....	28
2.2.4.1 Agisoft Metashape	28
2.2.4.2 Lage- und Höhenbezug	31
2.2.4.3 SAPOS-Zugang	32

2.2.4.4	Python Module	33
2.2.4.5	XMP-Metadaten	35
2.2.4.6	WebGIS (Geoportal)	36
2.3	Vorbereitende Maßnahmen.....	37
2.3.1	Ground Control Points	37
2.3.1.1	Anforderungen	37
2.3.1.2	Konstruktion einer Sprühschablone.....	38
2.3.1.3	Erstellung der GCPs	41
2.3.2	Flugplanung und -durchführung.....	44
2.3.2.1	Wetterbedingungen	45
2.3.2.2	Missionen	45
2.4	Automatisieren des Auswerteprozesses.....	48
2.4.1	Prozessübersicht.....	49
2.4.2	Die Hauptroutine Teil I	50
2.4.3	Implementierung der Hough Circle Transform Methode.....	55
2.4.3.1	Prozessübersicht	55
2.4.3.2	Beschreibung	56
2.4.4	Die Hauptroutine Teil II	66
2.4.5	Upload in den Cloudspeicher	68
3	Ergebnisse	69
3.1	Ergebnis Befliegung 1 (02.02.2023)	74
3.1.1	Orthomosaik und DEM	74
3.1.2	Erkennungsrate	75
3.1.3	Positionsgenauigkeit.....	76
3.2	Ergebnis Befliegung 2 (09.02.2023)	77
3.2.1	Orthomosaik und DEM	77
3.2.2	Erkennungsrate	78
3.2.3	Positionsgenauigkeit.....	80
3.3	Ergebnis Befliegung 3 (16.02.2023)	82
3.3.1	Orthomosaik und DEM	82
3.3.2	Erkennungsrate	83
3.3.3	Positionsgenauigkeit.....	85
3.4	Ergebnis Befliegung 4 (23.02.2023)	86
3.4.1	Orthomosaik und DEM	86

3.4.2 Erkennungsrate	87
3.4.3 Positionsgenauigkeit.....	88
3.5 Zusammenfassung	89
3.5.1 Erkennungsraten	89
3.5.2 Positionsgenauigkeiten	90
4 Diskussion	92
4.1 Erkennungsrate	93
4.2 Positionsgenauigkeit.....	95
5 Schlussfolgerung und Ausblick	101
6 Literaturliste.....	106
7 Anhang	110
7.1 Main.py	110
7.2 Set_Markers_HC.py.....	113
7.3 Beispiel XMP Metadaten	118
7.4 Ground Control Points.....	122
7.5 Checkpoints	123
7.6 Verwendete Software	124
7.7 Metashape Protokolle.....	125

Abkürzungsverzeichnis

bspw.	beispielsweise
ca.	circa
z.B.	zum Beispiel
vgl.	Vergleiche
o.g.	oben genannt
GCP	Ground Control Point
EPSG	European Petroleum Survey Group Geodesy
CRS	Coordinate Reference System
HT	Hough Transformation
HCT	Hough Circle Transformation
KI	künstliche Intelligenz
GNSS	Global Navigation Sattelite System
RTK	Real Time Kinematic
PPK	Post-Processing Kinematic
BB	Bounding Box
RMSE	Root Mean Square Error

Abbildungsverzeichnis

	<u>Seite</u>
Abbildung 1-1: Flächenplan Berlin TXL (GmbH 2023)	1
Abbildung 1-2: Eine DJI Matrice 300 RTK auf dem ehemaligen Vorfeld	3
Abbildung 1-3: Screenshot des Geoportals mit Vermessungsdaten (Bestandsplan) und Orthofoto im Hintergrund	5
Abbildung 1-4: Verteilung der aufgenommenen Fotos einer Gesamtbefliegung. Jeder Punkt stellt die Position eines Fotos dar.....	6
Abbildung 1-5: GCP in Form einer Luftbildplatte	7
Abbildung 1-6: Luftbildplatte neben einem kreisförmigen GCP aus 120m Flughöhe (vergrößerte Darstellung)	8
Abbildung 2-1: vereinfachte schematische Darstellung von Real Time Kinematik über SAPOS	15
Abbildung 2-2: Grafische Darstellung der Methode aus US-Patent Nr. 3,069,654 von Paul Hough	20
Abbildung 2-3: Darstellung der zwei Phasen von 21HT (Yuen, Princen et al. 1990)	23
Abbildung 2-4: Einer der Beiden DJI Matrice 300 RTK im Einsatz	24
Abbildung 2-5: Darstellung eines 3-Achsen-Gimbals am Beispiel DJI ZENMUSE Z30.....	25
Abbildung 2-6: Darstellung des Leica Zeno FLX100 auf einer Baustelle ...	27
Abbildung 2-7: Aktivieren des XMP-Metadaten Imports in Metashape.....	29
Abbildung 2-8: grafischer Überblick des brandenburgischen SAPOS- Referenzstationsnetzes (LGB 2021)	32
Abbildung 2-9: Auszug eines XMP-Metadatensatzes mit RTK- Korrekturwerten sowie Neigungswinkeln der Kamera bzw. des Gimbals.....	35
Abbildung 2-10: Screenshot des Geoportals mit eingeblendetem Orthofoto	36
Abbildung 2-11: Die selbstkonstruierte Sprühschablone im Einsatz. Hier befindet sich der Lotstab des GNSS-Messgerätes bereits in der Zentriervorrichtung.	40

Abbildung 2-12: Übersicht der angelegten Ground Control Points und des Umrings	41
Abbildung 2-13: Trocknung der Fläche	42
Abbildung 2-14: Einmessen des neuen GCP's mit per GNSS-Messgerät ..	43
Abbildung 2-15: Resultat des ersten GCPs	44
Abbildung 2-16: Screenshot des Flugmusters von Mission 1	46
Abbildung 2-17: Screenshot des Flugmusters von Mission 2	46
Abbildung 2-18: Screenshot des Flugmusters von Mission 3	47
Abbildung 2-19: Screenshot des Flugmusters von Mission 4	47
Abbildung 2-20: vereinfachte Darstellung des gesamten Auswertungsprozesses	49
Abbildung 2-21: vereinfachte Darstellung des Ablaufs der implementierten HCT-Methode	55
Abbildung 2-22: Marker vor der Korrektur (links) und Marker nach der Korrektur (rechts)	56
Abbildung 2-23: vereinfachte Darstellung des Upscalings	60
Abbildung 2-24: Erkannter GCP mit Mittelpunkt (grün) und Position vor Korrektur (rot)	65
Abbildung 3-1: Beispiel eines durch HCT erkannten GCPs. Die gelben Elemente wurden zur Veranschaulichung in Nachhinein hinzugefügt.	70
Abbildung 3-2: Luftbildplatte zur Vermarkung eines Checkpoints	72
Abbildung 3-3: kartografische Darstellung der angelegten Checkpoints ..	72
Abbildung 3-4: Erstelltes Orthomosaik vom 02.02.2023 (GSD 1,37cm) ..	74
Abbildung 3-5: Erstelltes digitales Höhenmodell vom 02.02.2023	74
Abbildung 3-6: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur	76
Abbildung 3-7: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint	76
Abbildung 3-8: Erstelltes Orthomosaik vom 09.02.2023 (GSD 1,37cm) ..	77
Abbildung 3-9: Erstelltes digitales Höhenmodell vom 09.02.2023	77

Abbildungsverzeichnis	X
<hr/>	
Abbildung 3-10: teilweise vom Schnee verdeckter GCP	79
Abbildung 3-11: von einer dünnen Schneedecke verdeckter GCP	79
Abbildung 3-12: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur	80
Abbildung 3-13: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint	80
Abbildung 3-14: Erstelltes Orthomosaik vom 16.02.2023 (GSD 1,37cm)	82
Abbildung 3-15: Erstelltes digitales Höhenmodell vom 16.02.2023	82
Abbildung 3-16: GCP erkannt, obwohl teilweise von Fahrzeug verdeckt ..	84
Abbildung 3-17: Der Schachtdeckel stellt für HCT eine Herausforderung dar	84
Abbildung 3-18: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur	85
Abbildung 3-19: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint	85
Abbildung 3-20: Erstelltes Orthomosaik vom 23.02.2023 (GSD 1,37cm)	86
Abbildung 3-21: Erstelltes digitales Höhenmodell vom 23.02.2023	86
Abbildung 3-22: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur	88
Abbildung 3-23: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint	88
Abbildung 3-24: Visualisierung der Erkennungsraten aller GCPs je Befliegung	89
Abbildung 3-25: Visualisierung der Gesamtfehlerwerte je Befliegung unterteilt nach Lage und Höhe	90
Abbildung 3-26: Visualisierung der summierten Fehlerwerte je Checkpoint unterteilt nach Lage und Höhe	91
Abbildung 3-27: Darstellung der mittleren Fehlerwerte der Checkpoints vor und nach der Optimierung durch GCPs	91
Abbildung 4-1: erkannter GCP auf Schachtdeckel	94
Abbildung 4-2: Erkannter GCP 3 bei schlechten Lichtverhältnissen	94

Abbildung 4-3: Vergleich von Ausschnitten aus Oblique-Aufnahmen mit einem Aufnahmewinkel von 45° bei ca. 90m Flughöhe. Der linke Ausschnitt befindet sich am oberen Bildrand und weist dadurch eine Höhere Verzerrung als der rechte Bildausschnitt auf, welcher sich eher am unteren Bildrand befindet. Zudem besteht die Darstellung aufgrund der Entfernung aus weniger Pixeln.	97
Abbildung 4-4: Verkleinerte Originalaufnahmen zu Abbildung 4-3.....	97
Abbildung 4-5: Beispiel Vergleich Befliegung 1 (links) mit Befliegung 4 (rechts) bei einem Maßstab von 1:20 über SWIPE-Tool in ArcGIS Pro. Ein minimaler Versatz ist erkennbar.	98
Abbildung 4-6: Beispiel Vergleich Befliegung 1 (oben) mit Befliegung 4 (unten) bei einem Maßstab von 1:200 über das SWIPE-Tool in ArcGIS Pro. Bei diesem Maßstab ist kein Versatz erkennbar.....	99
Abbildung 4-7: Darstellung von graphischem Bestandsplan über Orthomosaik von Befliegung 4	99
Abbildung 4-8: Auszug aus dem Metashape Protokoll mit Angaben zu den Positionsgenauigkeiten der Fotos.....	100
Abbildung 5-1: Programmablauf in der Windows Kommandozeile	104

Tabellenverzeichnis

Tabelle 1: Eckdaten der Mission und Flugparameter	45
Tabelle 2: Auflistung der Erkennungsraten von Flug1	75
Tabelle 3: Auflistung der Erkennungsraten von Flug2	78
Tabelle 4: Auflistung der Erkennungsraten von Flug3	83
Tabelle 5: Auflistung der Erkennungsraten von Flug4	87
Tabelle 6: Erkennungsrate je Befliegung.....	89

1 Einleitung

1.1 Der ehemalige Flughafen Berlin Tegel

Der ehemalige Flughafen Tegel macht Platz für ein modernes Berlin. Auf dem 500 ha großen Areal entstehen ein Forschungs- und Industriepark für urbane Technologien: Berlin TXL – The Urban Tech Republic und ein neues Wohnviertel: das Schumacher Quartier; zudem ein Landschaftsraum, der von der Firma Grün Berlin entwickelt wird.

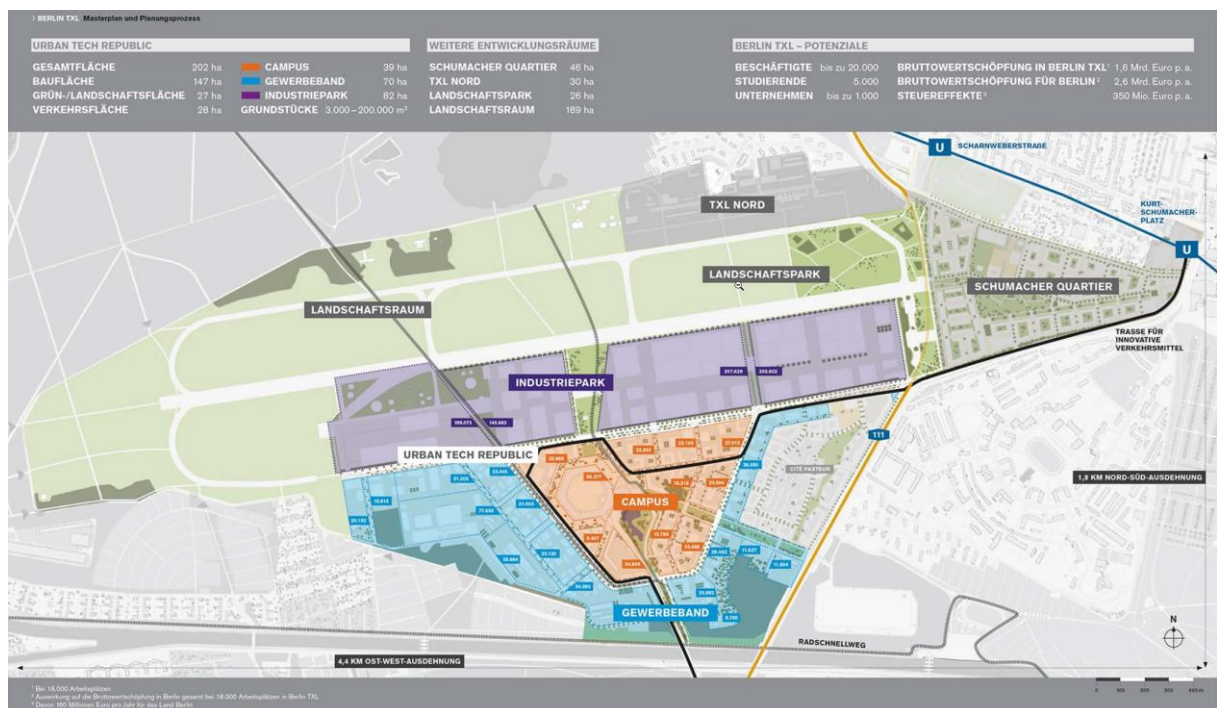


Abbildung 1-1: Flächenplan Berlin TXL (GmbH 2023)

In der Urban Tech Republic werden bis zu 1.000 große und kleinere Unternehmen mit 20.000 Beschäftigten forschen, entwickeln und produzieren und mehr als 2.500 Studierende werden mit der Berliner Hochschule für Technik in das ehemalige Terminalgebäude einziehen. Insgesamt sollen rund 5.000 Studierende den Campus Berlin TXL besiedeln. Im Fokus von Berlin TXL steht, was die wachsenden Metropolen des 21. Jahrhunderts am Leben erhält: der effiziente Einsatz von Energie, nachhaltiges Bauen, umweltschonende Mobilität, Recycling, die vernetzte Steuerung von Systemen, sauberes Wasser und der Einsatz neuer Materialien. Berlin TXL – The Urban

Tech Republic ist ein derzeit in Europa, wenn nicht weltweit, einzigartiges Vorhaben.

In der Nachbarschaft werden künftig diese neuen Entwürfe für das Leben in der Stadt der Zukunft greifbar sein: Im Schumacher Quartier entstehen über 5.000 Wohnungen für mehr als 10.000 Menschen in einem lebendigen, städtischen Quartier mit Kitas, Schulen und Einkaufsmöglichkeiten. Hier werden fortschrittliche Lösungen für die klimaneutrale Energieversorgung und hohe Energiestandards ebenso wie neue Mobilitätsmodelle aufgezeigt. Technologien, die nebenan – in der Urban Tech Republic – erforscht und entwickelt werden. Für die benachbarten Quartiere Cité Pasteur und TXL Nord sind weitere 4.000 Wohnungen geplant. Mit der Entwicklung und dem Management von Berlin TXL – The Urban Tech Republic und des Schumacher Quartiers hat das Land Berlin die Tegel Projekt GmbH beauftragt. Das landeseigene Unternehmen befasst sich u. a. mit den Planungen für den Hochbau und die technische, energetische und verkehrliche Infrastruktur, dem Bau- und Standortmanagement sowie mit der Flächenvermarktung und Kommunikation des Projektes in der Öffentlichkeit. Die Geländeübergabe an die Tegel Projekt GmbH erfolgte im August 2021. Nach Abschluss der vorbereitenden Maßnahmen wurden 2022 die Tiefbau- und Sanierungsarbeiten begonnen. Die Fertigstellung des 1. Bauabschnitts der Urban Tech Republic und des Schumacher Quartiers ist für 2027 geplant – so wie die mehrheitliche Vollendung der Gebäudesanierungen. (GmbH 2023)

1.2 Monitoring und Prozessunterstützung durch Drohnen

Die Aufgaben und Prozesse des Projektes werden von Beginn an durch Drohnentechnologie unterstützt. Vor allem die schnelle, genaue und vergleichsweise kostengünstige Erhebung von räumlichen Daten steht im Mittelpunkt des Interesses.



Abbildung 1-2: Eine DJI Matrice 300 RTK auf dem ehemaligen Vorfeld

Der Standardanwendungsfall ist die wöchentliche Bildbefliegung des Gesamtareals. Aus den gewonnenen Daten, ca. 10.000 Fotos, werden mittels Photogrammetrie und Computer Vision digitale Höhenmodelle (DEM) sowie digitale Orthofotos (DOP) erstellt. Durch den Einsatz von Drohnen besteht die Möglichkeit, diese detaillierten Aufnahmen des ca. 500 Hektar großen Areals in kurzer Zeit und mit vergleichsweise wenig Aufwand zu erzeugen. Die traditionelle Luftbildbefliegung mit einem konventionellen Flugzeug wäre im Vergleich dazu erfahrungsgemäß deutlich aufwendiger und wirtschaftlich nicht darstellbar. (Christian Martens 2023)

Die beiden wöchentlich erzeugten Produkte (DEM/DOP) und viele weitere projektrelevante Daten mit Raumbezug werden sämtlichen Kolleg*innen und vielen Projektbeteiligten direkt als Geodatendienst (WMTS) und zusätz-

lich einem Geoportal (siehe 2.2.4.6) zur Verfügung gestellt. Bei Bedarf können die Daten auch direkt als Download oder auf einem physischen Datenträger übergeben werden.

Insbesondere die Orthofotos decken mittlerweile eine Vielzahl von Anwendungsfällen ab. Im Bereich Facility Management (Liegenschaftsverwaltung) können Beschädigungen an Flächen und Gebäuden erkannt und behoben werden. Weiterhin kann die Durchführung fremdvergebener Dienstleistungen wie Grünpflege, Graufächenreinigung sowie Winterdienst jederzeit nachvollzogen werden. Im Zusammenhang mit GIS-gestützten Prozessen dienen die Fotos als Grundlage zum Flächenmanagement auf dem Areal. So können beispielsweise Mietflächen oder Baustelleneinrichtungsflächen in Lage und Ausdehnung mit wenigen Klicks validiert und gleichzeitig eine Beweissicherung (Vorher-Nachher) durchgeführt werden. Im Bereich Umweltschutz wird mithilfe von KI (Künstlicher Intelligenz) ein Floramonitoring zur Erkennung und Beobachtung invasiver Pflanzenarten durchgeführt. Auch für die Planung und den Bau erlangen die Orthofotos immer mehr Relevanz. Sie werden als Grundlage für langfristige Planungen wie den Landschaftspark verwendet, oder auch für die Planung von temporären Baustraßen, Trassen und weitere. Für die Kontrolle und Plausibilisierung von Bestandsplänen aus der Vermessung stellen die Fotos ebenfalls eine sinnvolle Grundlage dar. Mit ihrer Hilfe konnten zahlreiche Dokumentationslücken erkannt und gefüllt werden. So zeigt Abbildung 1-3 das Beispiel einer undokumentierten Sanierung eines Benzinabscheiders durch Überlagerung der Vermessungsdaten und eines Orthofotos. Diese Auflistung erhebt keinen Anspruch auf Vollständigkeit und erweitert sich stetig.



Abbildung 1-3: Screenshot des Geoportals mit Vermessungsdaten (Bestandsplan) und Orthofoto im Hintergrund

Mit der steigenden Akzeptanz und Nutzung dieser Daten sowie darauf aufbauender Prozesse ergibt sich gleichzeitig eine immer größere Verantwortung für die Datenerhebung. So muss nicht nur die regelmäßige (wöchentliche) Aktualisierung der Daten erfolgen, auch die Qualität der Daten muss dabei einem gewissen Standard entsprechen. Dies gilt nicht nur für die Datenerfassung im Felde, sondern auch für die nachgelagerte Datenauswertung.

Aufgrund der Datenmengen nimmt die Auswertung relativ viel Zeit in Anspruch. Der Grund dafür sind zum einen lange Berechnungszeiten (einzelne Teilschritte benötigen teilweise über 24h) und zum anderen Leerlaufzeiten zwischen den Arbeitsschritten im Auswerteprozess, da die Berechnung eines Teilschrittes selten innerhalb der Regelarbeitszeiten abgeschlossen ist. Die Befliegungen finden aus organisatorischen Gründen immer donnerstags statt. Dies hat zur Folge, dass viele Arbeitsschritte auf das darauffolgende Wochenende fallen. Zudem müssen einzelne Arbeitsschritte für einen nahtlosen Übergang teilweise nachts oder zu anderen unpässlichen Zeiten

durchgeführt werden. Weil dies nicht sehr praktikabel ist, ergab sich in der Praxis eine Gesamtprozessdauer von ca. 6 Tagen. Daher fiel die Entscheidung, eine Automatisierung des Auswertungsprozesses anzustreben, nicht schwer. Die erhofften Vorteile sind eine Verkürzung der Prozessdauer sowie die Vermeidung von menschlichen Fehlern, durch beispielsweise falsche Parametrierung.

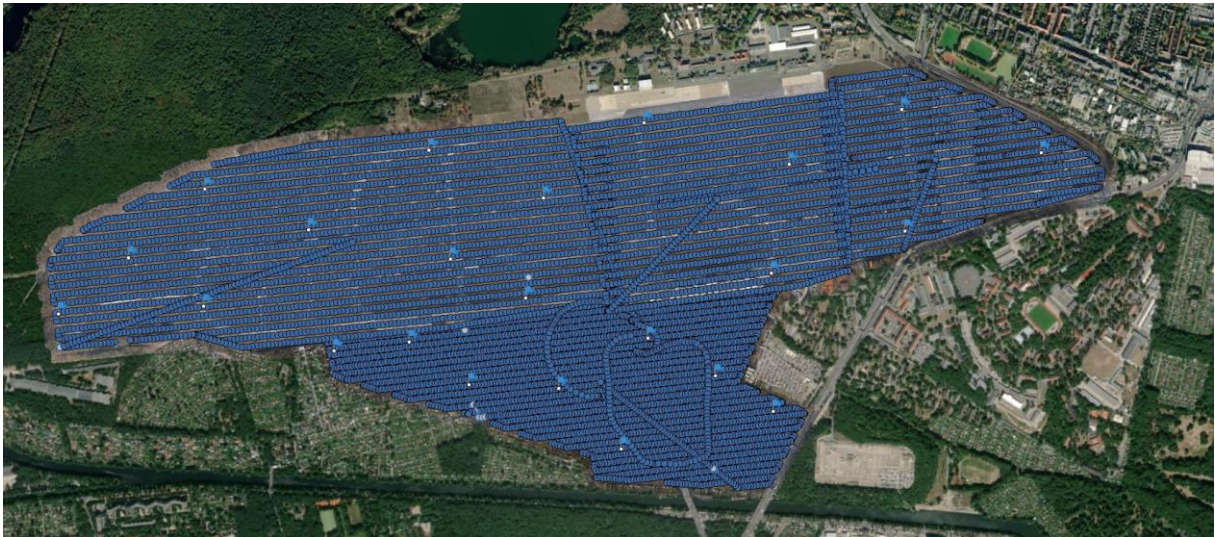


Abbildung 1-4: Verteilung der aufgenommenen Fotos einer Gesamtbefliegung. Jeder Punkt stellt die Position eines Fotos dar.

Die Automatisierung des Auswertungsprozesses soll Gegenstand dieser Arbeit. Ein besonderer Fokus fällt dabei auf die automatische Erkennung von Ground Control Points (GCPs), da die Datenqualität, wie bereits thematisiert, einen hohen Stellenwert einnimmt. Die GCPs werden benötigt, da trotz des Einsatzes von Real Time Kinematik (siehe 2.1.1.1) keine konstanten und genauen Positionsgenauigkeiten erzielt werden konnten. Auch (Lindstaedt and Kersten 2018) kommen zu dem Schluss, dass die Nutzung von Passpunkten nach wie vor einen hohen Stellenwert hat. Durch eine Kreuzbefliegung könnten die Ergebnisse signifikant verbessert werden (Kersten, Schlömer et al. 2020). Dies ist jedoch im Rahmen dieses Projektes durch die daraus resultierende Verdopplung von Befliegungszeit und den erzeugten Datenmengen aus Ressourcengründen keine Option. Die Bildflüge beschränken sich daher auf Streifenbefliegungen, sowie auf einigen

Aufnahmen in Schrägperspektive (siehe 2.3.2.2) und bedürfen der Unterstützung durch Passpunkte bzw. GCPs. Bedingt durch die Projektgröße hat sich eine räumlich gleichmäßig verteilte Anzahl von ca. 25 GCPs als ideal herausgestellt. Damit stellt auch der Ausfall von 1-2 GCPs kein unmittelbares Problem dar. Das manuelle Referenzieren der Passpunkte ist für den Bearbeiter die zeitaufwändigste Angelegenheit. Durch die Überlappung der Befliegung ist jeder Punkt auf 10 -20 einzelnen Fotos dargestellt und dementsprechend zu referenzieren. Bei 25 Passpunkten müssen somit in jedem Auswertungsprozess um die 400 Fotos abgearbeitet werden, was einen gemessenen Arbeitsaufwand von knapp 1,5h darstellt. Die Automatisierung dieses Vorgangs ist der Kernpunkt dieser Arbeit und letztendlich auch der Schlüssel für die Automatisierung des gesamten Auswerteprozesses.

Das Design der GCPs spielt eine wichtige Rolle und ist ebenfalls ein Gegenstand dieser Arbeit. Zu Beginn des Projektes wurde auf 50cm x 50cm große Luftbildplatten aus Kunststoff zurückgegriffen (siehe Abbildung 1-5). Diese Platten sind durch die Kontrastfarben sehr gut händisch referenzierbar (siehe Abbildung 1-6).



Abbildung 1-5: GCP in Form einer Luftbildplatte

Ein großer Nachteil der Luftbildplatten ist deren Beständigkeit. Viele Platten sind in kurzer Zeit durch Wind beschädigt worden oder sind sogar vollkommen verschwunden. Da die Platten zur Verankerung in unbefestigtem Untergrund gedacht sind, sind einige Exemplare auch der Grünpflege zum Opfer gefallen.

Die Grundlage für die Automatisierung der GCP-Erkennung ist eine entsprechende Bildanalyse und darauf aufbauend die Erkennung einer charakteristischen Geometrie. Durchsucht man die wissenschaftliche Literatur nach dieser Thematik, so stößt man immer wieder auf die sogenannte *Hough Circle Transform* – Methode (HCT). Diese Methode stammt aus dem Bereich der Computer Vision und dient allein dazu, Kreisförmige Objekte auf Rasterdaten zu erkennen und zu lokalisieren. Nach weiteren Recherchen wurde erfolgreich ein Prototyp, basierend auf eigenen Luftbildern, zur Erkennung von bestehenden kreisrunden Farbmarkierungen und anderen kreisförmigen Objekten auf dem Areal durchgeführt. Damit stand fest, dass die Luftbildplatten durch kreisrunde Farbmarkierungen ersetzt werden können, welche die Basis zur automatischen Erkennung bzw. Referenzierung von GCPs in der Auswertesoftware bilden. Nähere Ausführungen zur Entwicklung und Funktionsweise der HTC erfolgt im Kapitel 2.1.3. Das Thema *Ground Control Points* wird in den Kapiteln 2.1.2 und 2.3.1 weiter vertieft. Die folgende Abbildung zeigt die beiden beschriebenen Vermarktungsarten im unmittelbaren Vergleich:



Abbildung 1-6: Luftbildplatte neben einem kreisförmigen GCP aus 120m Flughöhe (vergrößerte Darstellung)

1.3 Stand der Forschung

Unmanned Aircraft Systems (UAS), die auch als Drohnen, Unmanned Aerial Vehicles (UAVs) und Remotely Piloted Aircraft (RPA) bezeichnet werden, haben einen langen militärischen Stammbaum, der ihr seit langem anerkanntes Potenzial zur Unterstützung der Kriegsführung widerspiegelt (Watts, Ambrosia et al. 2012).

Dass der Einsatz dieser Geräte nicht mehr länger ausschließlich der Kriegsführung vorbehalten ist, wurde bereits vor langer Zeit erkannt. Nach Nex and Remondino (2014) sind UAV-Plattformen heutzutage eine wertvolle Datenquelle für Inspektions-, Überwachungs-, Mapping- und 3D-Modellierungsprobleme. Sie können beispielsweise in der Fernerkundung eingesetzt werden. Diese wird traditionell mit dem Einsatz von Satelliten oder bemannten Flugsystemen, welche mit einer Vielzahl von Sensoren ausgestattet sind, in Verbindung gebracht (Pajares 2015).

Durch die rasante technische Weiterentwicklung in den letzten Jahren haben sich die Größe und das Gewicht dieser Sensoren bedeutsam verringert, sodass UAV-Plattformen dem Anwendungszweck entsprechend ausgestattet werden können. Die Fluggeräte werden unter anderem mit kostengünstigen Präzisionssensoren wie Inertial Motion Units (IMU), Gyroskopen und Global Positioning System (GPS)-Modulen zur automatischen Erkennung der eigenen Ausrichtung und Position ausgestattet (Siebert and Teizer 2014). Weiterhin ist das amerikanische GPS bei weitem nicht mehr das einzige Global Navigation Satellite System (GNSS). Die Systeme GLONASS, BeiDou und Galileo sind nunmehr ebenfalls im Einsatz und können laut Pan, Zhang et al. (2019) in kombinierter Nutzung die Signalverfügbarkeit sowie die Positionsgenauigkeit signifikant verbessern.

In Verbindung mit dem sogenannten RTK (Real Time Kinematic), einer Technik für den simultanen Empfang von GNSS-Signalen, kann die Genauigkeitssteigerung von UAV-Bildflügen nochmals erhöht werden. (Przybilla,

Reuber et al. 2015). Damit ist die Geodatenerfassung auf UAV-Basis mittlerweile salonfähig und es existieren zahlreiche Machbarkeitsstudien und praktische Anwendungsszenarien. So beweisen Mancini, Dubbini et al. (2013), dass eine schnelle, kostengünstige und hochautomatisierte Methode, welche 3D-Informationen aus unstrukturierten Luftbildern erzeugen kann, mittels UAV umsetzbar und gleichzeitig mit der Messgenauigkeit des Terrestrischen Laserscannings (TLS) komparabel ist. Kersten, Grahlmann et al. (2020) belegen in einem Pilotprojekt, dass die UAV-basierte Bildbefliegung gegenüber klassischer Vermessung eine zeitsparende Alternative zur Bestandsdatenerfassung in anforderungsentsprechender Genauigkeit darstellt. Kersten and Lindstaedt (2017) erläutern in einem Fachbeitrag über 4 verschiedene Projektbeispiele, dass nicht nur die Fluggeräte selbst sehr leistungsfähig sind. Auch die auf dem Markt erhältlichen Softwarepakete bieten dem Anwender von der Flugplanung, über die Durchführung bis hin zur Auswertung und Prozessierung der erfassten Daten sehr viele Möglichkeiten und erlauben darüber hinaus auch einen recht hohen Grad der Automation von Arbeitsschritten. Nichtsdestotrotz sind tiefgreifende Fachkenntnisse nach wie vor wünschenswert bis erforderlich, um ein qualitativ hochwertiges Produkt zu erzeugen. Lindstaedt and Kersten (2018) belegen, wie wichtig Passpunkte und deren optimale Verteilung im Objektraum für eine genaue Bildorientierung und 3D-Punktbestimmung durch Bündelblockausgleichung nach wie vor sind. Die geodätische Bestimmung und die Verteilung dieser Passpunkte sollte möglichst genau mit professionellem Vermessungsgerät und dem dazugehörigen Sachverstand durchgeführt werden. Der Einsatz von Passpunkten bzw. GCPs im Rahmen dieser Arbeit wurde im vorherigen Kapitel erläutert. Der besseren Übersicht halber wurde der Forschungsstand zu dieser Methode zusammen mit der technischen Erklärung im Kapitel 2.1.3 dargestellt.

1.4 Forschungsfrage

Die vorliegende Masterarbeit hat das Ziel zu demonstrieren, wie mit Hilfe von Hough Circle Transform eine Methode zur automatischen Erkennung von Ground Control Points (GCPs) in UAV-erfassten Bilddaten realisiert werden kann.

Ein weiterer Schwerpunkt liegt in der Anwendung von Open Source und freien Programmbibliotheken, um die Implementierung der entwickelten Methode in beliebigen Photogrammetrie-Anwendungen zu ermöglichen, insofern diese über eine geeignete Programmierschnittstelle verfügen.

Die entwickelte Methode soll als zentraler Bestandteil eines vollautomatischen Prozesses zur Verarbeitung von UAV-Bilddaten zur Erstellung eines georeferenzierten Orthomosaik in Vermesserqualität dienen, welche hier durch eine Lagegenauigkeit von mindestens 3 cm und einer Höhengenaugigkeit von mindestens 5 Zentimetern charakterisiert ist.

Die Evaluierung der erreichten Positionsgenauigkeit der entwickelten Methode erfolgt anhand von Analysen der erzielten Ergebnisse im Vergleich zu unabhängig vermessenen Referenzdaten (Checkpoints) im Forschungsgebiet.

2 Methodik

2.1 Grundlagen

2.1.1 GNSS (Satellitennavigation)

Die satellitengestützte Positionsbestimmung ist im Rahmen dieser Thesis eine Kernkomponente, da diese Technik in mehreren verwendeten Geräten zur Anwendung kommt. Dazu zählen die eingesetzten Quadrocopter sowie deren Fernsteuerungen und auch das GNSS-Messgerät (siehe 0), welches zur Einmessung der Ground Control Points verwendet wurde. Im Folgenden wird eine kurze Übersicht zur Funktionsweise der Satellitennavigation gegeben.

Satellitennavigationssysteme (SATNAV) sind weltraumbasierte Navigationshilfen, die aus einem Netzwerk von Satelliten bestehen, welche in einer Umlaufbahn um die Erde kreisen. Diese Systeme bieten oft eine sehr hohe Genauigkeit, sind jedoch auf Sichtlinien beschränkt, da die Signale von Satelliten empfangen werden müssen, die sich über dem Horizont befinden. Die meisten SATNAV-Systeme arbeiten mit relativ kurzen Wellenlängen, was zu einer hohen Genauigkeit führt, aber auch bedeutet, dass der Empfänger innerhalb der Sichtlinie der Satelliten bleiben muss (Kaplan and Hegarty 2017).

Heutzutage existieren zahlreiche SATNAV-Systeme auf der ganzen Welt. Einige operieren global und andere bieten nur innerhalb einer bestimmten Region ihre Dienste an. Der Begriff *Global Navigation Satellite System* (GNSS) definiert die Sammlung aller SATNAV-Systeme und ihrer Ergänzungen. Die momentan wichtigsten GNSS-Systeme sind das US-amerikanische *Global Positioning System* (GPS¹), das russische *GLobal Navigation Satellite System* (GLONASS²), das chinesische *BeiDou Navigation Satellite System*

¹ <https://www.gps.gov/>

² https://glonass-iac.ru/en/about_glonass/

(BDS³), das europäische *Galileo*⁴ System, das indische *Navigation with Indian Constellation* (NavIC⁵) und das japanische *Quasi-Zenith Satellite System* (QZSS⁶).

Alle GNSS-Systeme stellen genaue Positionsinformationen in drei Dimensionen sowie Geschwindigkeitsinformationen bereit und synchronisieren Zeitinformationen mit dem Coordinated Universal Time (UTC)-Zeitstandard. Die GNSS-Systeme bestehen aus einem Satellitensystem, einem Bodenkontroll- und Überwachungsnetzwerk sowie Benutzergeräten. Das Satellitensystem umfasst normalerweise 24 oder mehr Satelliten in mittleren Erdumlaufbahnen, angeordnet in 3 oder 6 Orbitalplänen mit jeweils vier oder mehr Satelliten pro Ebene. Die GNSS-Systeme arbeiten mit dem Prinzip der Einweg-Laufzeitmessung. Jeder Satellit sendet Signale aus, bei denen das Ranging-Code-Element auf einer gemeinsamen Zeitskala präzise synchronisiert ist. Die Navigationssignale geben an, wo sich der Satellit zum Zeitpunkt der Signalübertragung befindet, während der Ranging-Code es dem Empfänger ermöglicht, die Signalübertragungszeit zu bestimmen und damit die Satelliten-Benutzer-Entfernung zu berechnen. Der Empfänger benötigt auch eine Uhr, um die Laufzeitmessungen durchzuführen. Vier Satelliten sind erforderlich, um die dreidimensionale Position und die Geschwindigkeit des Benutzers sowie den Zeitunterschied zwischen der Benutzeruhr und der Satellitenuhr zu bestimmen (Kaplan and Hegarty 2017).

Inzwischen nutzen auch kommerzielle Empfangsgeräte Messungen von mehreren GNSS-Konstellationen, um eine präzise Positions-, Geschwindigkeits- und Zeitbestimmung zu ermöglichen. Dies stellt sicher, dass eine Signalverfügbarkeit gewährleistet ist, falls Probleme mit einem oder mehreren GNSS-Systemen auftreten und gewährleistet darüber hinaus eine höhere Verfügbarkeit bzw. Abdeckung (Montenbruck, Steigenberger et al. 2017, Pan, Zhang et al. 2019).

³ <http://en.beidou.gov.cn/>

⁴ <https://www.usegalileo.eu/EN/>

⁵ <https://www.isro.gov.in/SatelliteNavigationServices.html>

⁶ <https://qzss.go.jp/en/>

2.1.1.1 RTK (Real Time Kinematic)

Real Time Kinematic (RTK) ist eine Methode zur präzisen Bestimmung der Position eines Objekts mithilfe von GNSS. Im Gegensatz zur herkömmlichen GNSS-Positionierung, die eine Genauigkeit von einigen Metern bis zu einem Dutzend Metern erreicht, kann RTK eine Genauigkeit von wenigen Zentimetern oder sogar Millimetern erreichen.

Die Funktionsweise von RTK beruht auf der Messung der Phasenverschiebung von GNSS-Signalen zwischen einem Basis- und einem Rover-Empfänger über zwei Frequenzen (L1 und L2). Der Basisempfänger, oft auch Referenzstation genannt, verfolgt die Satellitensignale und sendet die gemessenen Phasenverschiebungen an den Rover-Empfänger, der die Differenz zwischen den Phasenverschiebungen misst. Diese Differenz wird dann verwendet, um die Position des Rovers relativ zur Position des Basisempfängers mit hoher Genauigkeit zu bestimmen (Montenbruck, Steigenberger et al. 2017). Als Alternative zur Referenzstation kann auch die Verbindung zu einem Satellitenpositionierungsdienst über NTRIP (Network Transport of RTCM via Internet Protocol⁷) hergestellt werden. Neben den Zugangsdaten zum Satellitenpositionierungsdienst wird eine permanente Internetverbindung benötigt (Przybilla and Baeumker 2020). Diese alternative Variante wurde auch im Rahmen der vorliegenden Arbeit verwendet. Zum Einsatz kam hier der SAPOS-Dienst des Landes Brandenburg (siehe: 2.2.4.1.3 und Abbildung 2-1).

⁷ Siehe: <https://igs.bkg.bund.de/ntrip/>

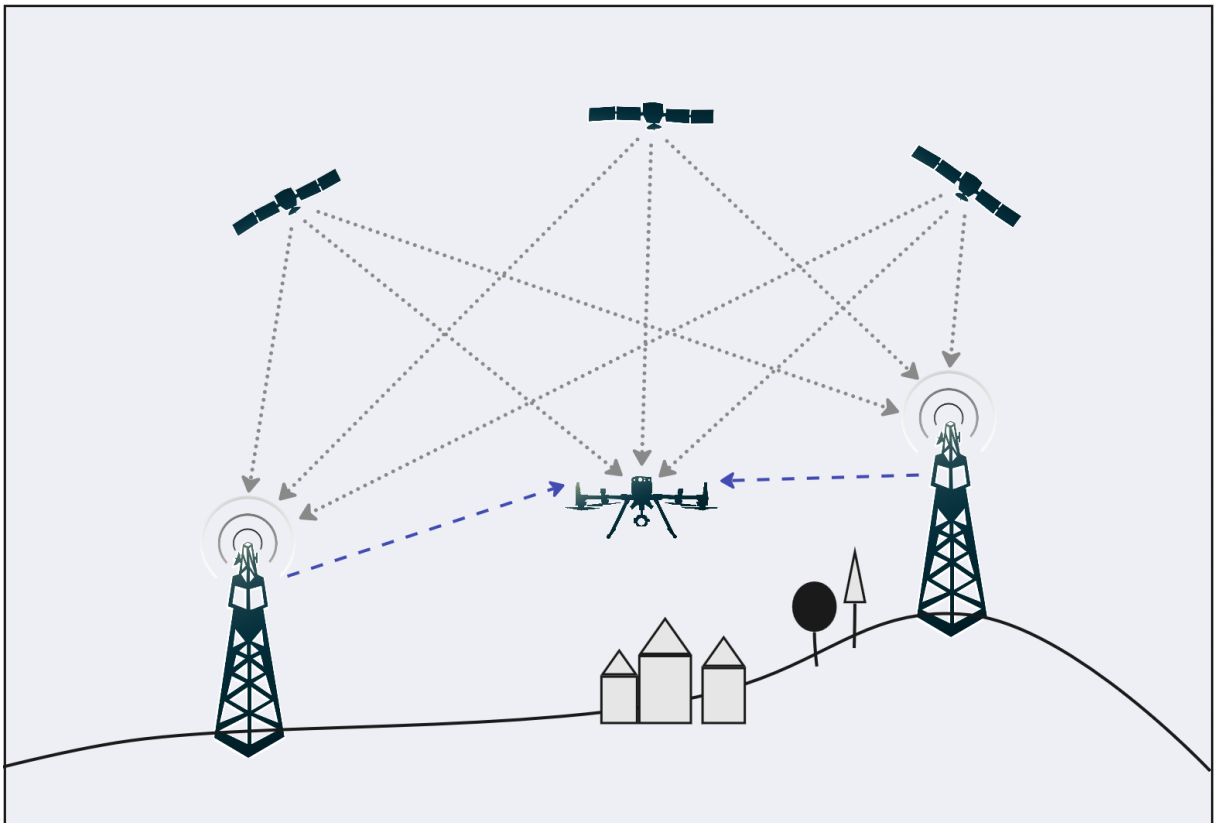


Abbildung 2-1: vereinfachte schematische Darstellung von Real Time Kinematik über SAPOS

Ein wichtiger Vorteil von RTK gegenüber herkömmlicher GNSS-Positionierung ist die höhere Genauigkeit, die durch die Verwendung von Phasenmessungen erreicht wird. Wübbena, Bagge et al. (2001) belegen, dass abhängig von der Qualität und Ausstattung des Rovers, sowie der Messparameter und Umweltbedingungen, Positionsbestimmungen mit einer Genauigkeit kleiner 1cm realisiert werden können. Ein weiterer Vorteil von RTK ist die Fähigkeit, Echtzeit-Korrekturdaten zu verwenden. Dadurch ist es möglich, die Position des Rovers mit sehr hoher Genauigkeit in Echtzeit zu bestimmen und zu verfolgen, was in vielen Anwendungen, wie z.B. der Landvermessung (Pirti, Arslan et al. 2009), dem Straßenbau (Halbrügge and Johanning 2015) und der automatisierten Landwirtschaft (Allred, Wishart et al. 2018), von entscheidender Bedeutung ist (Montenbruck, Steigenberger et al. 2017).

Auch kommerzielle UAVs können heutzutage mit RTK ausgestattet werden und profitieren davon in doppelter Hinsicht.

Einerseits ist die Positionierung des Gerätes selbst dadurch präziser, was sich positiv auf die Flugeigenschaften auswirkt. So kann beispielsweise die

Landung eines Quadropters nahezu exakt auf dem ursprünglichen Startpunkt erfolgen. Auch die Einhaltung virtueller geografischer Grenzen (Geofencing) wird dadurch verbessert und es ist sogar möglich mehrere UAVs im Verbund zu choreographieren, wie Ang, Dong et al. (2018) demonstrieren.

Andererseits profitieren auch die angebrachten Payloads der UAVs von RTK. So werden die Korrekturdaten bei aktuellen Geräten, wie beispielsweise des in diesem Projekt eingesetzten Modells *DJI Matrice 300 RTK* (siehe 2.2.1) oder *der DJI Phantom 4 RTK*, in Echtzeit mit den montierten Payloads synchronisiert. Das bedeutet, dass z.B. die Metadaten (siehe 2.2.4.5) von aufgenommenen Fotos (Kamera-Payload) direkt mit den präziseren bzw. korrigierten Positionsdaten angereichert werden (Przybilla and Baeumker 2020).

Diese Tatsache wirkt sich positiv auf die Auswertung, sowie die Weiterverarbeitung der Messdaten und damit letztendlich auch die Qualität der daraus generierten Produkte wie Höhenmodelle oder Orthofotos aus (Štroner, Urban et al. 2021).

Ein weiterer positiver Effekt von RTK betrifft die Verwendung von Ground Control Points bei UAV-gestützten Bildbefliegungen, wie sie auch im Zuge dieser Arbeit durchgeführt wurden. In diesem Zusammenhang belegen zahlreiche Studien (James, Robson et al. 2017, Kersten, Schlömer et al. 2020, Štroner, Urban et al. 2021, Peppas, Morelli et al. 2022), dass die verwendete Anzahl von GCP's signifikant verringert oder teilweise sogar darauf verzichtet werden kann, um eine Lage- und Höhengenaugkeit im einstelligen Zentimeterbereich zu erhalten.

Weitere Ausführungen zum Thema GCP's und deren besondere Relevanz in Bezug auf diese Arbeit folgen in den Kapiteln **2.1.2** sowie **2.3.1**.

2.1.2 Photogrammetrie (Structure from Motion)

Photogrammetrie ist ein Verfahren zur Erstellung von dreidimensionalen Modellen von Objekten oder Landschaften durch die Analyse von Fotos oder Bildern. Das Verfahren umfasst die Analyse von geometrischen Eigenschaften von Fotos und die Anwendung von mathematischen Algorithmen zur Rekonstruktion von 3D-Modellen. Photogrammetrie wird häufig in der Architektur, im Bauwesen, in der Landwirtschaft, in der Archäologie und in der Geologie eingesetzt (Kraus 2007).

Die Funktionsweise von Photogrammetrie basiert auf der Annahme, dass alle Objekte in der realen Welt eine einzigartige geometrische Struktur haben, die durch Fotos erfasst werden kann. Die Analyse der geometrischen Eigenschaften von Fotos, wie die Perspektive, die Entfernung und die Verzerrung, ermöglicht es, die Position, Form und Größe von Objekten zu bestimmen. Mathematische Algorithmen werden dann verwendet, um aus den erfassten Daten ein 3D-Modell zu erstellen (Kraus 2007).

Für die Erstellung von 3D-Modellen aus den Fotos werden spezielle Softwarelösungen verwendet. Einige bekannte Softwarelösungen sind das in dieser Arbeit eingesetzte *Agisoft Metashape*, sowie *Pix4Dmapper*, *Reality-Capture* oder *ArcGIS Reality*.

Die Arbeitsweise der genannten Softwarelösungen basiert in der Regel auf ein und denselben Grundsätzen bzw. Methoden der Photogrammetrie, welche teilweise mit Techniken aus der Computer Vision (Förstner and Wrobel 2016) ergänzt werden. nun nachfolgend anhand eines vereinfachten Verfahrensablaufes, mit dem Ziel der Erstellung eines Orthomosaiks, skizziert werden:

1. Aerotriangulation:

Die Aerotriangulation ist ein wichtiger Schritt in der Photogrammetrie, der die äußere Orientierung der Kamera in Bezug auf das Objekt ermittelt. Dabei wird die relative Position der Kamera zu Punkten auf

dem Boden berechnet, indem die Bündelblockausgleichung angewendet wird. Diese Methode verwendet sowohl Tie- als auch Keypoint-Informationen, um die Positionen der Kamera und der Punkte zu bestimmen. Tiefeninformationen werden aus der Verschiebung von Punkten in unterschiedlichen Bildern berechnet, während Keypoints durch Identifizierung von Merkmalen in den Bildern extrahiert werden. Die Innere Orientierung, die Eigenschaften der Kamera wie Brennweite und Verzerrungskoeffizienten beschreibt, wird ebenfalls während dieses Schritts bestimmt (Mikhail, Bethel et al. 2001).

2. Verbesserung der Inneren und Äußeren Orientierung durch Ground Control Points:

Die Verwendung von Referenzpunkten (GCPs) kann die Genauigkeit der äußeren und inneren Orientierung verbessern. GCPs sind bekannte Punkte auf dem Boden, die in den Bildern identifiziert und vermessen werden. Die Informationen aus den GCPs können zur Verbesserung der Schätzung der äußeren Orientierung verwendet werden, indem sie als zusätzliche Beobachtungen in die Bündelblockausgleichung aufgenommen werden. Die innere Orientierung kann ebenfalls durch GCPs verbessert werden, indem die Verzerrungskoeffizienten genauer bestimmt werden können (Wolf, Dewitt et al. 2014).

3. Erstellung einer 3D-Punktwolke:

Nach der Aerotriangulation kann eine hochaufgelöste 3D-Punktwolke durch die Verwendung von Depth Maps erstellt werden. Depth Maps sind 2D-Bilder, die die Entfernung jedes Pixels von der Kamera enthalten. Durch die Kombination von Depth Maps aus mehreren Blickwinkeln kann eine dichte Punktwolke erstellt werden, die die Geometrie des Objekts darstellt (James and Robson 2012).

4. Erstellung von Höhenmodellen:

Höhenmodelle können aus der 3D-Punktwolke erstellt werden, indem die Punkte in einem Raster angeordnet und interpoliert werden. Eine Möglichkeit, dies zu tun, ist die Verwendung von Triangulations-Algorithmen wie Delaunay-Triangulation oder Voronoi-Diagrammen. Eine weitere Möglichkeit ist die Verwendung von Interpolationsmethoden wie Kriging oder Splines. Das resultierende digitale Höhenmodell (DEM) kann zur Erstellung von digitalen Geländemodellen (DTMs) oder digitalen Oberflächenmodellen (DSMs) verwendet werden, welche wichtige Informationen für eine Vielzahl von Anwendungen wie Hydrologie, Geomorphologie und Kartographie liefern (Colomina and Molina 2014).

5. Erstellung von Orthomosaiken:

Orthomosaiken können aus Höhenmodellen erstellt werden, indem die Texturinformationen aus den Originalbildern auf die geometrisch korrigierte Oberfläche projiziert werden. Dies führt zu einer korrekten geometrischen Darstellung der Oberfläche, die für viele Anwendungen wie Kartierung, Landnutzungsbewertung und Umweltüberwachung nützlich ist (Remondino and El-Hakim 2006).

2.1.3 Hough Circle Transform

Die *Hough Transformation (HT)* ist eine Technik, welche zum Auffinden von Linien, Kreisen bzw. anderen simplen Formen in einem Bildraster entwickelt wurde. In der ursprünglichen Variante wurde diese Methode genutzt, um ein binäres Bild nach geraden Linien zu durchsuchen bzw. gerade Linien zu erkennen. Durch gezielte Weiterentwicklung konnte diese Erkennung auf weitere Formen ausgedehnt werden (Kaehler 2008). Die heutzutage verwendete *HT* war demzufolge keine Entwicklung, welche in einem einzigen Schritt durchgeführt wurde. Vielmehr waren viele einzelne Schritte und Verbesserungen nötig, um die Methode zu dem zu machen, was sie heute ist

(Hart 2009). Den Grundstein legte 1962 der Physiker P. V. C. Hough mit dem US-Patent Nr. 3,069,654 - *Method and Means For Recognizing Complex Patterns*. Die patentierte Erfindung beinhaltete eine Methode zur maschinellen Erkennung von komplexen Mustern und Linien in fotografischen oder anderen bildlichen Darstellungen. Der Hintergrund dieser Erfindung war zu jener Zeit die Untersuchung von subatomaren Teilchenbahnen in einer Blaskammer auf dem Gebiet der Kernphysik (Hough 1962). Auch viele Jahre später wird die HT-Methode beispielsweise durch Amram (2008) noch immer in modifizierter Form auf diesem Wissenschaftsgebiet angewandt, um Myonenspuren im *Large Hadron Collider* des CERN⁸ nachzuweisen (Hart 2009).

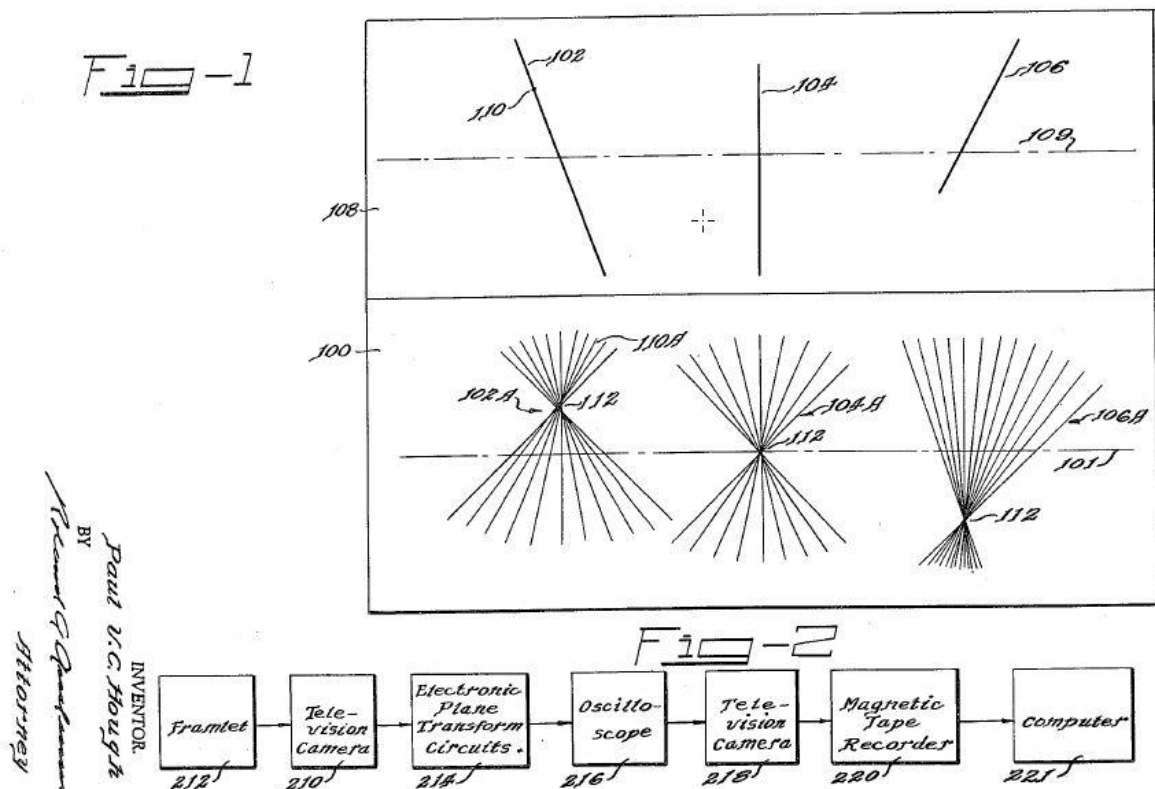


Abbildung 2-2: Grafische Darstellung der Methode aus US-Patent Nr. 3,069,654 von Paul Hough

Nach Illingworth and Kittler (1988) war die HT-Methode in Ihrer ursprünglichen Form nicht sehr beliebt und konnte zunächst nur selten eingesetzt werden, da sehr viele Berechnungen nötig waren und der Speicherbedarf

⁸ Siehe: <https://home.cern/science/accelerators/large-hadron-collider>

für die damalige Zeit zu hoch war. Doch im Laufe der Zeit konnte sich *HT* durch die Weiterentwicklung der Computerleistung, sowie durch die Optimierung der Methode selbst, immer mehr durchsetzen. So waren Shibata and Frei (1981) bereits dazu in der Lage, in Echtzeit Ziele auf Infrarotbildern zu Erkennen. Aber auch im Rahmen aktueller Forschungen in diversen Wissenschaftszweigen erfreut sich *HT* noch immer großer Beliebtheit. Im Bereich der Medizin verwenden Vijayarajeswari, Parthasarathy et al. (2019) unter anderem *HT* erfolgreich zur Früherkennung von Brustkrebs. Chen, Qiang et al. (2021) nutzen die Methode zur Extraktion von Navigationspfaden für die automatische Navigation von Agrarrobotern. Darüber hinaus kann die *HT*-Methode auch auf Videoinhalte angewendet werden. Dies belegen u.a. Iqbal, Iqbal et al. (2020), indem Sie mittels *HT* eine Linienerkennung als Grundlage für verschiedene hochrangige Aufgaben wie Objekt-, Anomalie- und Aktivitätserkennung vorstellen. Die Physiker Serra, Masotti et al. (2020) gehen noch einen Schritt weiter und kombinieren *HT* erfolgreich mit einer KI bzw. einem Neuronalen Netzwerk zur Analyse von Strömungen in passiv gekühlten Atomreaktoren auf Basis von Bläschenbildung.

Die Theorie der Hough Transformation kann zusammengefasst wie folgt formuliert werden:

Jeder Punkt in einem binären Bild kann Teil einer Menge von möglichen Linien sein. Durch die Parametrisierung jeder Linie mit einer Steigung a und einem y -Achsenabschnitt b wird ein Punkt im Originalbild in einen Ort von Punkten in der (a, b) -Ebene umgewandelt, der allen möglichen Linien entspricht, die durch diesen Punkt verlaufen. Durch die Umwandlung aller Pixel ungleich Null im Eingangsbild in solche Punktgruppen im Ausgangsbild und die Summierung aller Beiträge werden Linien, die im Eingangsbild vorhanden sind (d.h. in der (x, y) -Ebene), als lokale Maxima im Ausgangsbild (d.h. in der (a, b) -Ebene) erscheinen. Die (a, b) -Ebene wird aufgrund der Beiträge jedes Punktes oft als *Akkumulator-Array* bezeichnet (Kaehler 2008).

Um nun statt einfacher Linien die kreisförmigen Ground Control Points auf den Luftbildern zu erkennen, bedarf es einer Modifikation der klassischen

HT-Methode: die *Hough Circle Transform*-Methode (*HCT*). Diese Methode arbeitet ähnlich wie die *HT*, indem auch hier ein Akkumulator-Array verwendet wird, um die Anzahl von Linien für jede mögliche Kreisposition und -größe im Bild zu zählen (Kimme, Ballard et al. 1975).

Allerdings ist die *HCT* aufgrund der zusätzlichen Dimension des Kreisradius nicht so einfach wie die klassische *HT*. Wenn versucht wird, eine exakte Anwendung des Verfahrens durchzuführen, würde ein dreidimensionales Akkumulator-Volumen benötigt werden, was zu einem erhöhten Speicherbedarf und langsamerer Geschwindigkeit führen würde (Kaehler 2008). Um diese Herausforderung zu bewältigen, verwendet die Implementierung in der im Rahmen dieser Arbeit genutzten *OpenCV*-Bibliothek (siehe 2.2.4.1.3) die *Hough Gradienten Methode*, oder auch *2-1 Hough Transform (21HT)*. Diese Methode ist in der Lage, die Kreiskandidaten in einer zweidimensionalen Akkumulator-Ebene zu zählen, indem sie zunächst eine Kanten-Erkennungsphase durchführt, gefolgt von der Berechnung der lokalen Gradienten an jedem nicht-null-Punkt im Kantenbild (Yuen, Princen et al. 1990).

Für jeden Punkt ungleich Null wird die Linie, die durch den Gradienten angezeigt wird, von einem bestimmten Minimum bis zu einem bestimmten Maximum in der Akkumulator-Ebene inkrementiert. Die Position jedes Punktes ungleich Null im Kantenbild wird ebenfalls gespeichert. Anschließend werden aus den Kreiskandidaten in der Akkumulator-Ebene jene ausgewählt, die sowohl einen bestimmten Schwellenwert überschreiten als auch größer sind als ihre direkten Nachbarn. Die ausgewählten Kreiskandidaten werden nach ihren Akkumulator-Werten sortiert, wobei jene mit den meisten unterstützenden Kantenpixeln zuerst erscheinen. Für jeden ausgewählten Kandidaten werden dann alle nicht-null-Punkte nach ihrem Abstand zum Kandidaten sortiert. Ausgehend von den kleinsten Abständen wird ein einzelner Kreisradius ausgewählt, der am besten von den nicht-null-Punkten unterstützt wird. Der Kandidat wird beibehalten, wenn er ausreichend Unterstützung von den Punkten ungleich Null im Kantenbild hat und ausreichend weit von bereits ausgewählten Kreiszentren entfernt ist. (Hough 1962, Yuen, Princen et al. 1990, Rhody 2005, Kaehler 2008, Hart 2009).

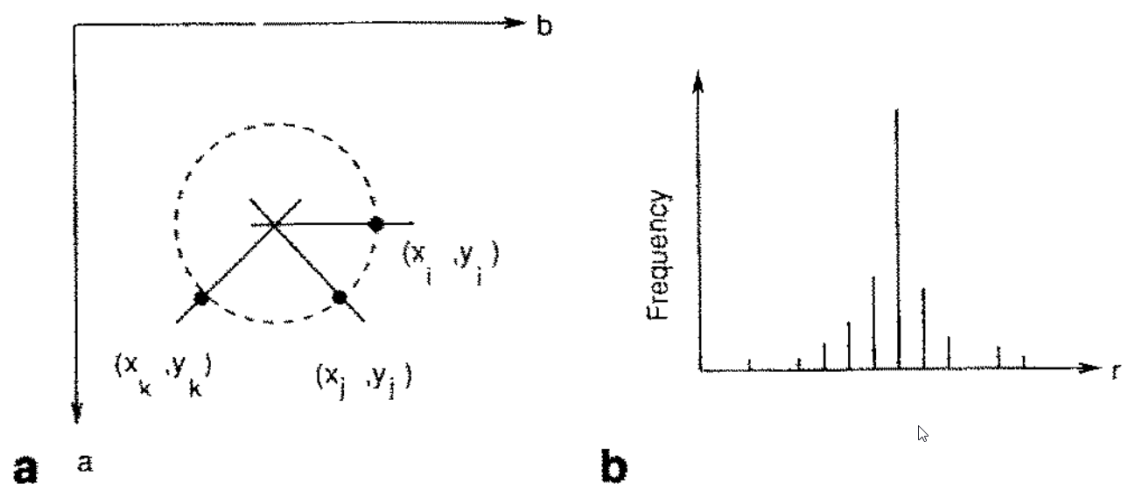


Figure 4. 21HT decomposes the circle detection problem into two stages. In (a) a transform is accumulated to find centre coordinates. In (b) a radius histogram is constructed for each candidate centre derived in (a)

Abbildung 2-3: Darstellung der zwei Phasen von 21HT (Yuen, Princen et al. 1990)

2.2 Eingesetzte Ausrüstung

2.2.1 UAV-Plattform (DJI M300 RTK)

Für die Befliegungen wurden zwei Modelle vom Typ *DJI Matrice 300 RTK (M300)* eingesetzt. Es handelt sich dabei um ein modulares Quadrocopter-System des Herstellers DJI, welches speziell für den kommerziellen Einsatz entwickelt wurde. Das System zeichnet sich vor allem durch seine Robustheit und Vielseitigkeit aus. So können je nach Anforderung bis zu 3 verschiedene Nutzlasten (engl.: Payloads) gleichzeitig an den Copter angebracht und verwendet werden. Dies schließt sowohl die speziell für diese Plattform entwickelten Payloads als auch spezielle Payloads von Drittanbietern mit ein. Die zulässige Betriebstemperatur liegt laut Hersteller im Bereich von -20°C bis 50°C . Zudem verfügt der Copter über eine IP45-Schutzklassifizierung gemäß Standard IEC 60529 (IEC 2013), sowie einem Windwiderstand von maximal 15m/s. Zusätzlich ist das Fluggerät mit einem integrierten RTK-Modul ausgestattet, welches präzise Flugrichtungsdaten bereitstellt und somit eine genauere Positionierung ermöglicht. (DJI 2020)

In diesem Projekt sind beide Copter jeweils mit der Photogrammetrie-Kamera *DJI Zenmuse P1* (vgl. 2.2.1.1) auf der Unterseite, sowie dem Fallschirmsystem *PRS-300* des Herstellers *AVSS* (vgl. 2.2.1.2) auf der Oberseite ausgerüstet.

Zur Steuerung der beiden Copter wurde jeweils ein *DJI Smart Controller Enterprise* eingesetzt. Diese Controller wurden speziell für die *M300* entwickelt und ist Teil der *M300*-Plattform. Auf den Controllern läuft ein Android-Betriebssystem und die *DJI Pilot 2* App zur Flugplanung und -durchführung (siehe 2.3.2). Beide Fernsteuerungen waren über einen WLAN-Hotspot permanent mit dem Internet verbunden, um die RTK-Korrekturdaten über *SAPOS* (siehe 2.2.4.1.3) zu empfangen und an die Drohnen zu übertragen.



Abbildung 2-4: Einer der Beiden DJI Matrice 300 RTK im Einsatz

2.2.1.1 Photogrammetrie-Kamera (DJI Zenmuse P1)

Die *Zenmuse P1* ist eine Vollformat-Kamera, die speziell für den photogrammetrischen Einsatz an der *M300* entwickelt wurde. Die Kamera verfügt über einen 45 Megapixel Vollformatsensor mit globalem mechanischem Verschluss, welcher ein konstantes Aufnahmeintervall von 0,7s ermöglicht. Die *P1* ist an einem durch drei Achsen stabilisierten Gimbal (siehe Abbildung 2-5) an der Drohne aufgehängt und mit einem Objektiv mit fester Brennweite (hier 35mm) ausgestattet. Die Aufnahmen der Kamera werden in Echtzeit durch die RTK-Funktion der *M300* korrigiert. Sollte im Aufnahmegebiet kein RTK verfügbar sein, so ist es im Nachhinein auch möglich die Korrektur der Aufnahmepositionen mittels PPK durchzuführen. (DJI 2021)

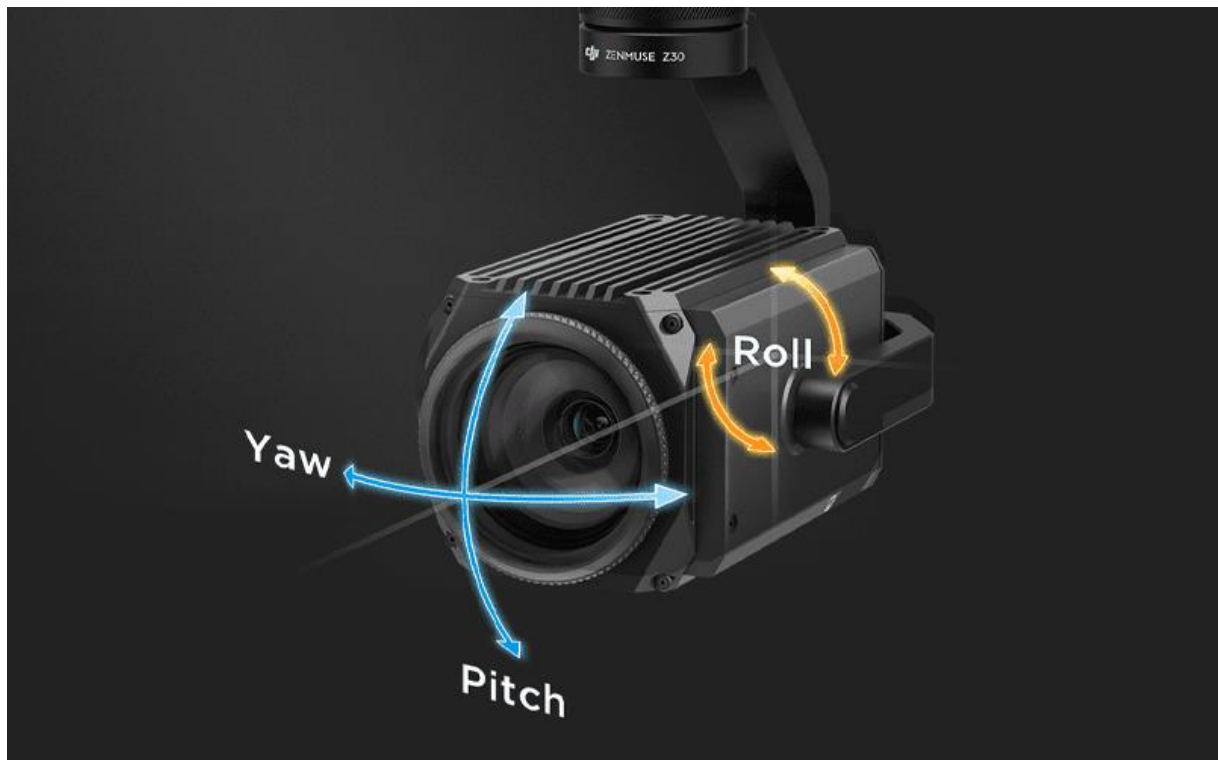


Abbildung 2-5: Darstellung eines 3-Achsen-Gimbals am Beispiel DJI ZENMUSE Z30⁹

⁹ Quelle: <https://frontierprecision.com/wp-content/uploads/DJI-Zenmuse-Z30-3.png>

2.2.1.2 Fallschirm (AVSS PRS-300)

Das Fallschirmsystem vom kanadischen Hersteller AVSS wurde speziell für die Matrice 300 konzipiert. Das System ist fest auf dem Copter installiert und zusätzlich über ein Kabel (OSDK Port¹⁰) mit der Bordelektronik verbunden. Im Falle einer Fehlfunktion oder dem Überschreiten eines Grenzwertes (beispielsweise Neigung $> 30^\circ$) werden die Motoren der Drohne gestoppt und der Schirm wird entfaltet. Zusätzlich hat der Pilot die Möglichkeit den Schirm über eine manuelle Auslösevorrichtung zu entfalten und ggf. die Motoren zu stoppen. (AVSS 2022)

2.2.2 GNSS-Messgerät (Leica Zeno FLX 100)

Zum Einmessen der Ground Control Points wurde das GNSS-Messgerät *Zeno FLX100* von der Firma *Leica* eingesetzt. Es handelt sich dabei um ein marktübliches Messgerät. Mit der RTK-Echtzeitkorrektur liefert das FLX100 eine horizontale Messgenauigkeit von 2 cm und eine vertikale Messgenauigkeit von 3 cm. Unterstützt werden die gängigen Satellitenpositionierungssysteme GPS¹¹, Glonass¹², BeiDou¹³, Galileo¹⁴, QZSS¹⁵ sowie SBAS¹⁶ auf insgesamt 186 Kanälen (AG 2020). Das Messgerät wurde über ein *LEICA ZENO TAB 2* angesteuert. Dabei handelt es sich um einen Tablet-Computer mit Android Betriebssystem und einem LTE¹⁷-Modem um die RTK-Korrekturdaten über SAPOS (siehe 2.2.4.2) zu empfangen und mit den Messwerten der Smartantenne zu synchronisieren.

¹⁰ siehe auch: <https://developer.dji.com/onboard-sdk/documentation/quickstart/device-connection.html>

¹¹ <https://www.gps.gov/>

¹² https://glonass-iac.ru/en/about_glonass/

¹³ <http://en.beidou.gov.cn/>

¹⁴ <https://www.usegalileo.eu/EN/>

¹⁵ <https://qzss.go.jp/en/>

¹⁶ <https://www.euspa.europa.eu/european-space/eu-space-programme/what-sbas>

¹⁷ <https://www.4g.de/>



Abbildung 2-6: Darstellung des Leica Zeno FLX100 auf einer Baustelle¹⁸

2.2.3 Prozessierungsrechner

Bei dem eingesetzten Prozessierungsrechner handelt es sich um einen, in Eigenregie zusammengestellten, Desktop-Computer mit handelsüblichen Hardware-Komponenten. Es wurde insbesondere auf eine leistungsfähige Kühlung geachtet, damit die Komponenten im Dauerbetrieb nicht überhitzen und eine möglichst hohe Lebensdauer besitzen. Für diesen Zweck wurden hochwertige Industrielüfter, sowie eine Wasserkühlung für den Prozessor verbaut. Bei den Hardware-Kernkomponenten handelt es sich um folgende:

- Prozessor: *AMD Ryzen 9 5950X* (16 CPU-Kerne)
- Grafikkarte: *NVIDIA GeForce RTX 3090* (24GB)
- Arbeitsspeicher: *Corsair 128 GB DDR4* (4*32GB)
- Motherboard: *Gigabyte Aorus Master X570S*
- Datenspeicher:
 - *Samsung SSD 980 PRO 2TB* (System)
 - ca. 50TB Festplattenspeicher (Archivspeicher)

Sämtliche Datenauswertungen wurden auf diesem Gerät durchgeführt. Perspektivisch ist die Ablösung dieses Systems durch eine Cloudlösung geplant.

¹⁸ Quelle: <https://leica-geosystems.com/de-de/products/gis-collectors/smart-antennas/leica-flx100>

2.2.4 Software

2.2.4.1 Agisoft Metashape

Als grundlegende Software für die photogrammetrische Datenauswertung wurde im Rahmen dieser Arbeit Agisoft Metashape eingesetzt. Diese Software ist für die Durchführung jedoch nicht obligatorisch und könnte auch durch eine andere proprietäre- oder Open Source Software ersetzt werden.

Agisoft Metashape ist eine Software, die digitale Bilder, wie z.B. Luftaufnahmen, Satellitenbilder oder Nahbereichsaufnahmen, photogrammetrisch verarbeitet. Die Software generiert 3D-Raumdaten, die für verschiedene Anwendungen, wie beispielsweise GIS-Anwendungen, kulturelle Dokumentationen oder visuelle Effekte, genutzt werden können.

Die Software verarbeitet Bilder von RGB-, Wärmebild- oder Multispektralkameras sowie strukturierten terrestrischen Laserscans und unstrukturierten Luftlaserscans und generiert daraus Raumdaten in Form von Punktwolken, texturierten polygonalen Modellen, georeferenzierten True Orthomosaiken und digitalen Höhen- bzw. Geländemodellen (DEMs/DTMs). Mithilfe von Post-Processing-Funktionen können Schatten und Texturartefakte aus den Modellen entfernt werden. Metashape nutzt digitale Photogrammetrie-Technik und Computer-Vision-Methoden, um eine intelligente und automatisierte Verarbeitung zu ermöglichen.

Sowohl Anfänger als auch Spezialisten können die Software nutzen und haben dabei volle Kontrolle über die Ergebnisgenauigkeit. Nach der Verarbeitung wird ein detaillierter Bericht erstellt.

(Agisoft-LLC 2023)

2.2.4.1.1 Arbeitsschritte in Metashape

Der allgemeine Arbeitsablauf von den Rohdaten (Fotos) bis hin zum Orthomosaik wurde unter 2.1.2 beschrieben. Eine detailliertere Beschreibung des erarbeiteten Workflows wird im Zuge der Prozessautomatisierung unter 2.4 ausgeführt.

2.2.4.1.2 Notwendige Programmeinstellungen

Um sämtliche Metadaten aus den aufgenommenen Fotos importieren und in den Auswerteprozess mit einbeziehen zu können, musste die Importfunktion für XMP-Metadaten in Metashape einmalig grundlegend aktiviert werden:

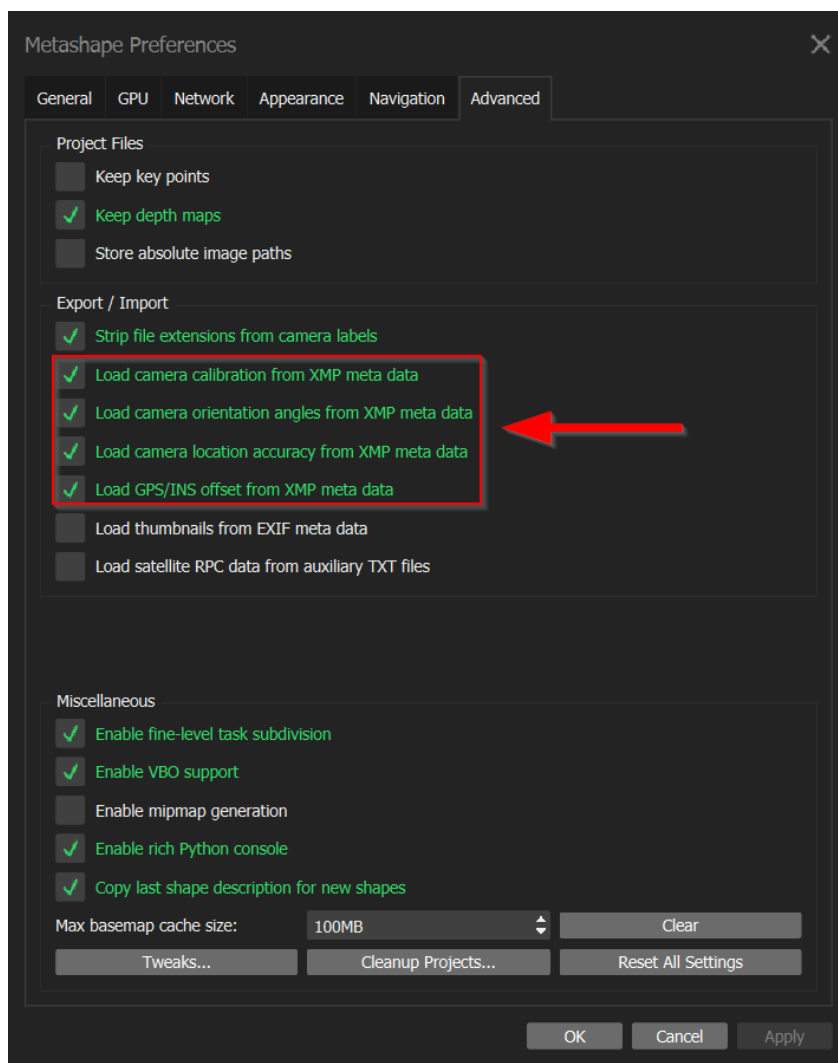


Abbildung 2-7: Aktivieren des XMP-Metadaten Imports in Metashape

2.2.4.1.3 Metashape Python API

Die Python API von Agisoft Metashape ermöglicht die programmatische Steuerung und Automatisierung von Aufgaben in der Software. Mit der API können Entwickler Skripte erstellen, die Aufgaben wie das Öffnen und Speichern von Projekten, die Verarbeitung von Fotos, die Erstellung von Punktwolken, die Erstellung von Texturen und die Erstellung von 3D-Modellen automatisieren.

Einige der wesentlichen Funktionalitäten der Python API für Agisoft Metashape sind:

- Zugriff auf Projektdaten: Die API ermöglicht es, Projekte zu öffnen, zu speichern und zu bearbeiten, sowie auf die Projektdaten wie Fotos, Punktwolken und 3D-Modelle zuzugreifen.
- Automatisierung von Verarbeitungsschritten: Mit der API können Entwickler Schritte wie die Verarbeitung von Fotos, die Erstellung von Punktwolken und die Erstellung von 3D-Modellen automatisieren.
- Erstellung von benutzerdefinierten Arbeitsabläufen: Mit der API können Entwickler benutzerdefinierte Arbeitsabläufe erstellen, die auf spezifische Anforderungen oder Workflows abgestimmt sind.
- Zugriff auf erweiterte Funktionen: Die API bietet Zugriff auf erweiterte Funktionen, die in der regulären Benutzeroberfläche von Agisoft Metashape nicht verfügbar sind.

2.2.4.2 Lage- und Höhenbezug

Als Lagebezugssystem kam sowohl für die Auswertung als auch für die finalen Produkte (DEM, Orthomosaik) das in Berlin amtliche System *ETRS89 / UTM zone 33N*¹⁹ zur Anwendung.

Das Deutsche *Haupthöhenetz 2016 (DHHN2016)*²⁰ wurde Höhenbezugssystem verwendet.

Da sich die durch RTK-GNSS erzeugten Koordinaten im geodätischen Referenzsystem *ETRS89*²¹ mit ellipsoidischen Höhen handelt, mussten diese zunächst transformiert werden.

Das *Leica FLX100* wurde bereits für Messungen im amtlichen Referenzsystem vorprogrammiert und führte die benötigten Transformationen direkt beim Speichern der Messpunkte aus.

Die Koordinaten der aufgenommenen Fotos der beiden DJI M300 RTK wurden im Zuge des automatisierten Auswerteprozesses (siehe 2.4.2) in Agisoft Metashape transformiert. Dafür musste zur Transformation von ellipsoidischen Höhen nach DHHN2016 der *German Combined Quasigeoid (GCG2016)*²² im GeoTIFF-Format beschafft und in Metashape implementiert²³ werden.

¹⁹ siehe: <https://epsg.io/25833>

²⁰ siehe: <https://epsg.io/7837>

²¹ siehe: <https://epsg.io/4258>

²² siehe: <http://gdz.bkg.bund.de/index.php/default/quasigeoid-der-bundesrepublik-deutschland-quasigeoid.html>

²³ siehe: <https://agisoft.freshdesk.com/support/solutions/articles/31000148332-how-to-use-height-above-geoid-for-the-coordinate-system>

2.2.4.3 SAPOS-Zugang

Die RTK-Verbindungen wurde im Rahmen dieser Arbeit über SAPOS-HEPS (Hochpräziser Echtzeit Positionierungs-Service) hergestellt.

Über diesen Service wird eine Positionsgenauigkeit von bis zu 1-2 cm in der Lage und 2-3 cm in der Höhe ermöglicht. Die Informationen werden über das Internet im RTCM-Format übertragen. Das Netz der Referenzstationen besteht aus insgesamt 43 verschiedenen Stationen in und um das Bundesland Brandenburg (siehe Abbildung 2-8). (LGB 2021)

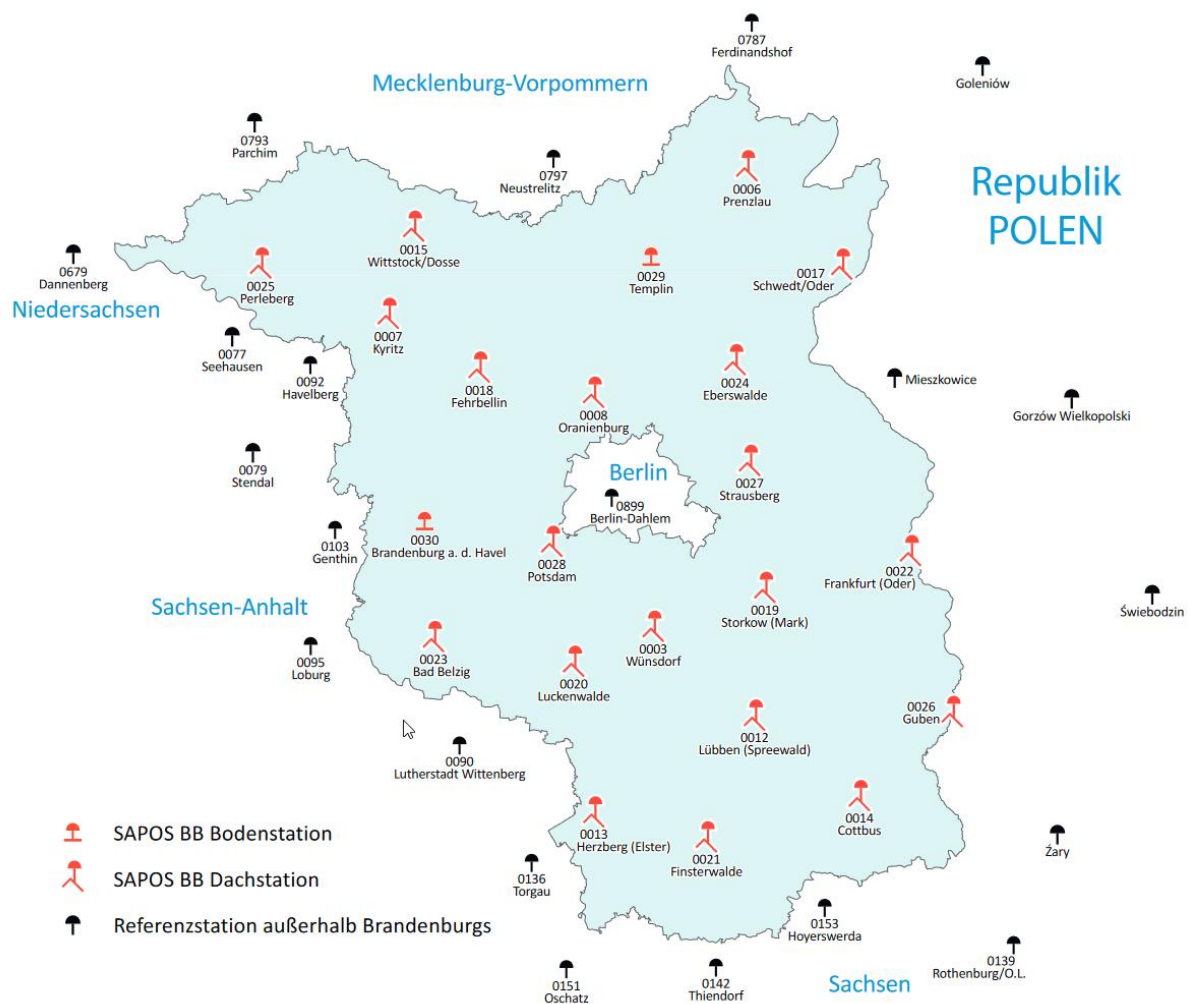


Abbildung 2-8: grafischer Überblick des brandenburgischen SAPOS-Referenzstationsnetzes (LGB 2021)

2.2.4.4 Python Module

Das Scripting wurde ausschließlich mithilfe der Programmiersprache Python in Version 3.8 durchgeführt. Bei Python handelt es sich um eine interpretierte, hochflexible und interdisziplinäre Programmiersprache mit starkem Fokus auf Lesbarkeit und Übersichtlichkeit. Sie ist verhältnismäßig einfach zu erlernen und wird oft für wissenschaftliche Anwendungen, Datenanalyse, Webentwicklung und maschinelles Lernen verwendet. (Python-Software-Foundation 2023)

Abgesehen von der API²⁴ der proprietären Software Agisoft Metashape, sind für Automatisierung des Auswerteprozesses, sowie im Besonderen für die GCP-Erkennung durch *Hough Circle Transform* weitere Module bzw. Bibliotheken zum Einsatz gekommen. Dabei wurde ausschließlich auf freie Komponenten zurückgegriffen, um die Methode zur GCP-Erkennung möglichst niederschwellig in Alternativenwendungen zu Agisoft Metashape implementieren zu können.

Im Folgenden eine kurze Aufstellung der verwendeten Komponenten, welche nicht Teil des Python Softwarepaketes sind:

- **OpenCV:**

Auch bekannt als Open Source Computer Vision Library, ist eine Software-Bibliothek für Computer Vision und maschinelles Lernen, die als Open-Source-Projekt verfügbar ist. Ihr Zweck besteht darin, eine standardisierte Infrastruktur für Computer-Vision-Anwendungen bereitzustellen und die Integration von maschinellem Wahrnehmungsvermögen in kommerzielle Produkte zu erleichtern. Durch die Verwendung der Apache-2-Lizenz²⁵ ermöglicht OpenCV auch Unternehmen

²⁴ API steht für *Application Programming Interface* und bezeichnet eine Programmierschnittstelle *Luber, t. u. D.-I. F. S. (2023, 08.03.2017). "Definition „Programmierschnittstelle“ Was ist eine API?" Retrieved 05.03.2023, 2023, from <https://www.dev-insider.de/was-ist-eine-api-a-583923/>.*

²⁵ siehe: Foundation, T. A. S. (2004). "Apache License, Version 2.0." Retrieved 05.03.2023, 2023, from <https://www.apache.org/licenses/LICENSE-2.0>.

den einfachen Zugriff auf den Code und dessen Modifikation, was Flexibilität und Anpassungsfähigkeit fördert. Zum aktuellen Zeitpunkt umfasst OpenCV mehr als 2500 Algorithmen. (OpenCV 2023)

OpenCV ist die Kernkomponente für die GCP-Erkennung in dieser Arbeit, da es unter anderem die Algorithmen für die *Hough Circle Detection* enthält. Die exakte Implementierung wird unter 2.4.3 beschrieben.

- **NumPy:**

Ein Open Source Projekt, welches speziell für numerische Berechnungen konzipiert wurde. NumPy steht für jeden Nutzer unter einer modifizierten BSD-Lizenz²⁶ zur Verfügung. Die exakte Implementierung wird unter 2.4.3 beschrieben.

²⁶ siehe: Project, T. L. I. (2005, 22.04.2005). "BSD License Definition." Retrieved 05.03.2023, 2023, from <http://www.lininfo.org/bsdlicense.html>.

2.2.4.5 XMP-Metadaten

Für sämtliche Fotos der DJI M300 RTK werden Metadaten in Form des EXIF- und XMP-Standards (ISO 2019) gespeichert. Im Zuge der Datenauswertungen wurden die XMP-Daten durch Metashape ausgelesen und entsprechend in den Algorithmen berücksichtigt. Der XMP-Standard bietet umfassendere Abbildungsmöglichkeiten als der weitverbreitete EXIF-Standard. Zu diesen zählen die in Metashape berücksichtigten Genauigkeitsangaben der Lage- bzw. Höhenkoordinaten nach der RTK-Korrektur sowie Angaben zu den Neigungswinkeln des Kamera-Gimbals. Der komplette XMP-Metadatenatz eines Fotos aus diesem Projekt kann im Anhang (siehe 7.3) eingesehen werden.

```
Gps Status : RTK
Altitude Type : RtkAlt
Absolute Altitude : +194.686
Relative Altitude : +120.007
Gimbal Roll Degree : +0.00
Gimbal Yaw Degree : -82.10
Gimbal Pitch Degree : -89.90
Flight Roll Degree : -0.40
Flight Yaw Degree : -76.00
Flight Pitch Degree : -10.30
Flight X Speed : 2.4
Flight Y Speed : -9.7
Flight Z Speed : 0.0
Cam Reverse : 0
Gimbal Reverse : 0
Self Data :
Rtk Flag : 50
Rtk Std Lon : 0.00978
Rtk Std Lat : 0.01409
Rtk Std Hgt : 0.02017
Rtk Diff Age : 1.80000
Surveying Mode : 1
UTC At Exposure : 2023:02:23 11:57:29.427597
Shutter Type : Mechanical
Shutter Count : 245841
Camera Serial Number : 3XMDJAT001325B
Drone Model : Matrice 300 RTK
```

Abbildung 2-9: Auszug eines XMP-Metadatenatzes mit RTK-Korrekturwerten sowie Neigungswinkeln der Kamera bzw. des Gimbals²⁷

²⁷ Die Auslesung der Metadaten erfolgte mithilfe der freien Software *ExifTool* by Phil Harvey Harvey, P. (2023). "ExifTool by Phil Harvey." Retrieved 05.03.2023, 2023, from <https://exiftool.org/>.

2.2.4.6 WebGIS (Geoportal)

Das Geoportal ist ein webbasiertes Kartenwerkzeug und zentraler Zugangsort für alle projektbezogenen Geodaten sowie weiteren Informationen, die einen räumlichen Bezug zum Projektgebiet haben. Das umfasst insbesondere Bestands- und Planungsdaten, ALKIS-Daten, die DOPs und DEMs, weitere anwendungsfallsspezifische Fachdaten, Sensordaten sowie offene Daten des Landes Berlin.

Die Geodaten werden als OGC-Dienste (WMS, WMTS, WFS) visualisiert. Das Geoportal stellt diverse Standard-Web-GIS-Funktionalitäten bereit. Die Möglichkeit zur webbasierten Dateneditierung ist geplant.

Die Zugangsverwaltung erfolgt dabei zentral über ein Identity Management System. Die Daten und Funktionalitäten werden über eine rollenbasierte Steuerung den genannten Zielgruppen zugänglich gemacht.

Sämtliche Komponenten des Geoportals basieren auf Open Source Technologie. Als Frontend kommt das sogenannte Masterportal²⁸ zum Einsatz, als Geodatenserver werden QGIS-Server²⁹ sowie Geoserver³⁰ eingesetzt.

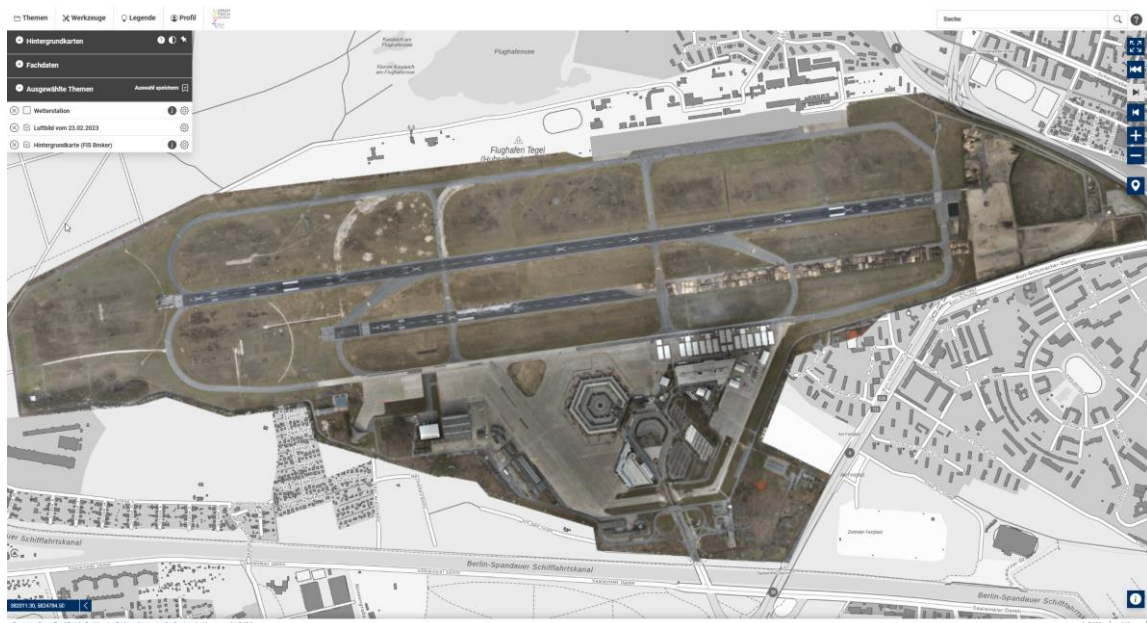


Abbildung 2-10: Screenshot des Geoportals mit eingeblendetem Orthofoto

²⁸ siehe: <https://www.masterportal.org/>

²⁹ siehe: https://docs.qgis.org/3.22/de/docs/server_manual/

³⁰ siehe: <https://geoserver.org/>

2.3 Vorbereitende Maßnahmen

2.3.1 Ground Control Points

2.3.1.1 Anforderungen

Um durch Hough Circle Transform effektiv erkannt zu werden und um deren Parametrierung zu vereinfachen, sollten alle Ground Control Points (GCP's) in der Örtlichkeit möglichst identisch beschaffen sein und die folgenden Kriterien erfüllen:

1. Die Größe der Kreise sollte immer identisch sein, da diese in der Parametrierung der Algorithmen berücksichtigt wird. Im Prototyp konnte mit einem Durchmesser von 30cm gute Ergebnisse erzielt werden, daher wurde dieser Wert als Größe festgelegt.
2. Es sollte sichergestellt werden, dass es sich um ideale Kreise handelt und nicht um ähnliche Geometrien wie beispielsweise Ellipsen.
3. Die Kreise sollten in einem einheitlichen Farbwert sein und sich möglichst kontrastreich von der Umgebung abheben. Im Falle dieser Arbeit hat sich der Autor bewusst für die Farbe Weiß entschieden, da die Verortung der GCPs im Forschungsgebiet ausschließlich auf befestigten Untergründen wie Asphalt und Beton durchgeführt wurde. Auf sehr hellen Untergründen könnte sich jedoch ein dunkler Farbton besser eignen. Diese Entscheidung ist somit stark von der Beschaffenheit des Einsatzgebietes abhängig und muss in der Parametrierung der Algorithmen definiert werden.
4. Es sollten sich keine weiteren kreisförmigen Elemente in unmittelbarer Umgebung eines GCPs befinden, da es dadurch möglicherweise zu Verwechslungen kommen kann. Die Folge wäre, dass die Algorithmen möglicherweise keine oder die falschen Ergebnisse liefern. Durch gezielte Parametrierung und Abfragen im Programmiercode, sowie unter Berücksichtigung der drei vorhergenannten Punkte, kann eine Verwechslung zumindest weitestgehend ausgeschlossen werden.

Weitere Anforderungen bestehen aus Sicht von Herrichtung und Wartung. Trotz sorgfältiger Planung der einzelnen Positionen durch Abgleich mit Abbruchplänen und Bauplanungen kann es immer wieder vorkommen, dass einzelne GCP's im Laufe der Zeit zerstört oder zumindest beschädigt werden. Daher sollten die Punkte möglichst unkompliziert, schnell und mit einfachen Mitteln erstellt, sowie gegebenenfalls ausgebessert werden können.

2.3.1.2 Konstruktion einer Sprühschablone

Um die unter 2.3.1.1 genannten Anforderungen an GCPs zu erfüllen, hat sich der Autor dieser Arbeit für die Konstruktion einer Sprühschablone zum Auftragen von Bodenmarkierfarbe entschieden. Die Schablone wurde aus Buchensperrholz mit einer Stärke von 8mm gefertigt. Der Ablauf der Konstruktion stellt sich wie folgt dar:

Für alle Arbeitsschritte wurde das Werkstück jederzeit mit Schraubzwingen an der Werkbank fixiert, um ein Verrutschen oder Verspringen des Werkstücks zu verhindern. Diese Maßnahme trug, neben dem Tragen einer Schutzbrille, eines Gehörschutzes, einer Atemschutzmaske (FFP2) sowie Arbeitshandschuhen zur Arbeitssicherheit bei und garantierte zudem präzises Arbeiten.

Die Sperrholzplatte wurde mittels einer Tauchsäge und einer zugehörigen Führungsschiene auf ein bündiges Zielformat von 60cm x 60cm zugeschnitten. Anschließend wurde auf der Platte der exakte Mittelpunkt mit einem Bleistift und der Führungsschiene angerissen. Dieser Mittelpunkt diente als Ausgangspunkt für den auszuschneidenden Kreis mit exakt 30cm Durchmesser. Das Ausschneiden des Kreises wurde durch die Verwendung einer Oberfräse und eines Fräszirkels, welcher im angerissenen Mittelpunkt der Platte fixiert wurde, durchgeführt. Dabei war zum einen auf die Einstellung des korrekten Radius und einer ausreichenden Fixierung zwischen Zirkel und Oberfräse zu achten, um den Radius auch beizubehalten. Zum anderen musste der auszufräsende Kreis aus Sicherheitsgründen mit 2 Senkkopfschrauben fixiert werden, um ein unkontrolliertes Rotieren oder Wegfliegen

am Ende der Fräsung zu unterbinden. Senkkopfschrauben waren notwendig, um die Rotation des Fräszirkels nicht zu behindern. Da die eingesetzte Oberfräse nicht über eine Tauchfunktion verfügte, wurde der Einstieg mit einem 10mm Holzbohrer gesetzt. Als Fräskopf kam ein 10mm Nutfräser zum Einsatz.

Nachdem das Loch gefräst wurde, musste eine abnehmbare Zentriervorrichtung als Positionierungshilfe für den GNSS-Lotstab angebracht werden. Mithilfe dieser Zentriervorrichtung wird die Einmessung des exakten Kreismittelpunktes eines jeden GCPs gewährleistet. Die Vorrichtung besteht aus einem einfachen Kantholz (unbekanntes Nadelholz) mit den Maßen 2cm x 4cm, welches mithilfe einer japanischen Zugsäge auf die benötigte Länge von 40cm zugesägt wurde. Da mit frischer Farbe gearbeitet werden sollte, wurde bewusst kein Schiebe- oder Klappmechanismus verwendet, um keine Farbe zu verschmieren. Stattdessen wurde eine abnehmbare Steckverbindung mit Holzdübeln realisiert. Dazu wurden auf den Hilfslinien, die bereits zur Bestimmung des Schablonenmittelpunktes angerissen wurden, gegenüberliegend Löcher mit einem 10mm Holzbohrer in die Schablone gebohrt. Ein weiteres 10mm Loch wurde in den Mittelpunkt gebohrt. Aus diesem Grund war es wichtig, den durch Schrauben fixierten ausgefrästen Kreis vorerst an Ort und Stelle zu belassen. Die Positionen der Bohrlöcher in der Schablone wurden anschließend mit Holzdübel-Zentrierspitzen auf das Kantholz übertragen. In die beiden äußeren Positionen wurden mithilfe des 10mm Holzbohrers und Holzleim Holzdübel gesetzt. Vorher wurden die Holzdübel mit der japanischen Zugsäge entsprechend auf Länge gekürzt, damit diese nach Aufstecken der Zentriervorrichtung nicht auf der Unterseite der Schablone hervorstehen. Schließlich wurde der Mittelpunkt mit einem 20mm Forstner Bohrer durchstoßen, um durch diesen den Lotstab führen zu können.

Zum Schluss wurden zwei Haltegriffe in Form von Kanthölzern angebracht. Diese wurden wieder mit der japanischen Zugsäge auf eine einheitliche Länge zugeschnitten. Die Positionierung erfolgte auch hier über die Hilfslinien zur Bestimmung des Schablonenmittelpunktes. Die Befestigung wurde

mit jeweils zwei einfachen Senkkopfschrauben von der Schablonenunterseite aus vorgenommen. Um keine Risse zu erzeugen, wurde mit einem 2mm Holzbohrer sowohl in der Schablone als auch in den Haltegriffen vorgebohrt. Zusätzlich wurden die Bohrlöcher an der Unterseite der Schablone mit einem Kegelsenker bearbeitet, damit die Schrauben keine Risse im Sperrholz erzeugen und auch nicht aus diesem hervorstehen. Letzteres hätte ein unerwünschtes Kippen der Schablone zur Folge. Sämtliche Bohrungen und Verschraubungen wurden mit einem handelsüblichen Akkuschauber durchgeführt.



Abbildung 2-11: Die selbstkonstruierte Sprühschablone im Einsatz. Hier befindet sich der Lotstab des GNSS-Messgerätes bereits in der Zentriervorrichtung.

2.3.1.3 Erstellung der GCPs

Für die Forschungsarbeit wurde in einem Arbeitsgang das gesamte GCP-Netz bestehend aus insgesamt 23 Punkten mit einem 2-Mann Team in der Örtlichkeit hergestellt. Die folgende Abbildung zeigt eine Lageübersicht der neu erstellten GCPs sowie den Umring des Orthomosaiks, welches letztendlich erstellt werden soll.

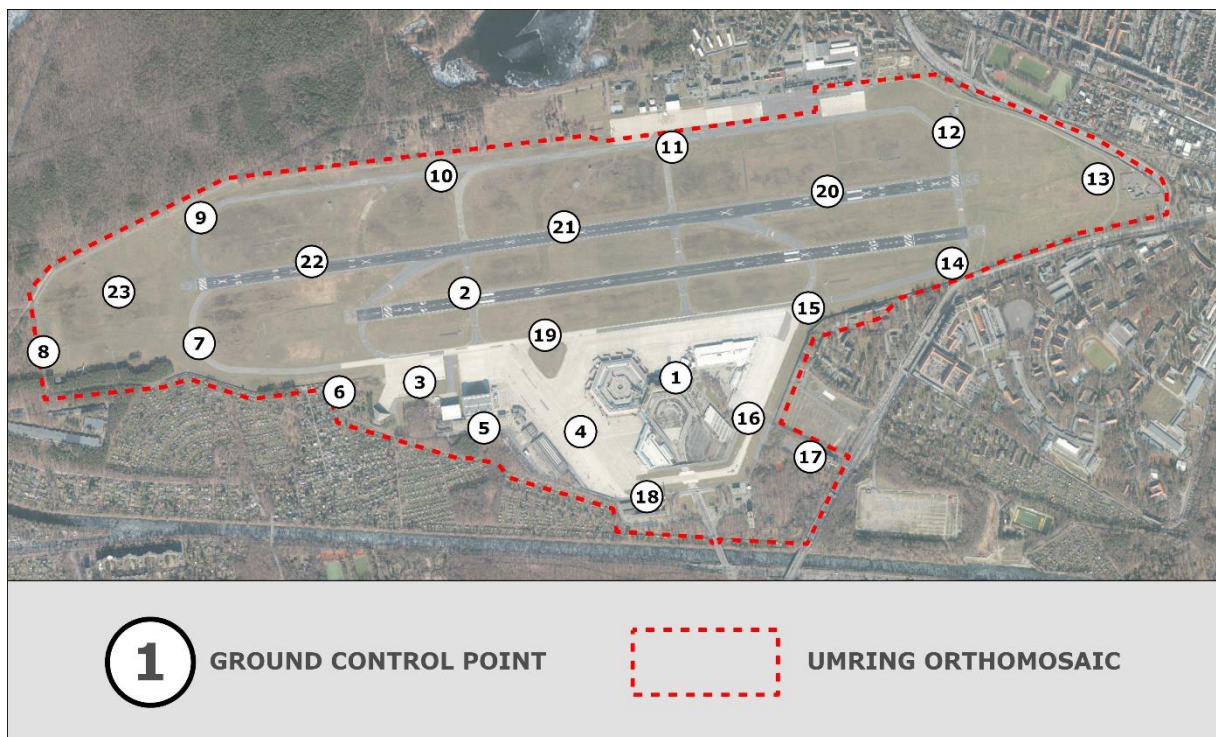


Abbildung 2-12: Übersicht der angelegten Ground Control Points und des Umrings

Das Prozedere zum Erstellen der GCPs wurde einheitlich durchgeführt und kann wie folgt beschrieben werden:

Zuerst wurde die Zielfläche mit einem Propangasbrenner getrocknet. Der Vorteil dieser Methode ist, dass der Luftstrom der Flamme wie ein Gebläse wirkt und die Fläche gleichzeitig von Schmutzpartikeln wie Staub oder Sand befreit. Dies unterstützt eine bessere Haftung und somit die Beständigkeit der Farbe. Zusätzlich wurde die Fläche erwärmt, was den Trocknungsprozess zusätzlich beschleunigte.



Abbildung 2-13: Trocknung der Fläche

Im nächsten Schritt wurde die Sprüschablone auf der getrockneten Fläche platziert und die Farbe gleichmäßig aufgetragen. Als Farbe kam die Sprüh-Bodenmarkierfarbe *SOPPEC TRACING plus* in weiß zum Einsatz, da diese in ihren Eigenschaften als besonders widerstandsfähig und witterungsbeständig beworben wird, sowie über eine kurze Trockenzeit von 10 bis 15 Minuten verfügt. (Peter Boels 2023)

An dieser Stelle sei erwähnt, dass auch jede andere Markierungsfarbe verwendet werden kann und dass der Autor keine Bewertung hinsichtlich Qualität oder besonders guter Eignung vornimmt. Das Produkt wurde rein zufällig anhand der oben genannten Eigenschaften ausgewählt.

Direkt nach dem Auftrag der Markierfarbe wurde die Zentriervorrichtung auf die Sprüschablone aufgesetzt und die Einmessung des Kreismittelpunktes mit dem GNSS-Messgerät durchgeführt.



Abbildung 2-14: Einmessen des neuen GCP's mit per GNSS-Messgerät

Eine stichprobenartige Kontrolle der neu erstellten GCPs am Folgetag zeigte, dass die Punkte einem ergiebigen Niederschlag in der Nacht schadlos standgehalten haben und optimal getrocknet zu sein schienen. Die Konturen waren immer noch scharf und die Farbe war nicht verlaufen.



Abbildung 2-15: Resultat des ersten GCPs

2.3.2 Flugplanung und -durchführung

Die Befliegungen des Areals wurden zeitgleich mit zwei Drohnen und demzufolge auch zwei Piloten durchgeführt. Durch diese Arbeitsteilung wurde die Gesamtzeit der Befliegungen im Gegensatz zu Befliegungen mit nur einer Drohne nahezu halbiert. In ca. 3,5 - 4 Stunden konnte der Gesamtprozess unter guten Bedingungen abgeschlossen werden. Dies beinhaltet neben den Befliegungen auch die Rüstzeiten.

2.3.2.1 Wetterbedingungen

Es existieren verschiedene Faktoren, welche sich negativ auf die Gesamtflugzeit auswirken können. So können beispielsweise Niederschläge wie Regen und Schnee oder auch tiefhängende Wolken sowie Bodennebel den Start der Befliegung hinauszögern, unterbrechen oder im schlimmsten Fall auch unmöglich machen. Zudem kann es bei hoher Luftfeuchtigkeit und Temperaturen nahe bzw. unterhalb des Gefrierpunktes zu Eisbildung an den Geräten und vor allem an den Rotoren kommen. Auch starke Winde bzw. Böen können die Befliegung erschweren oder verhindern.

2.3.2.2 Missionen

Das Forschungsgebiet wurde in vier verschiedene Sektoren unterteilt, welche jeweils durch eine Mission in der DJI Pilot2 App des Smartcontrollers abgebildet werden. Durch diese Aufteilung wurde gewährleistet, dass mit zwei Coptern gleichzeitig geflogen werden kann. Eine graphische Übersicht der Missionen wird in Abbildung 2-16 bis Abbildung 2-19 dargestellt. Eine Besonderheit ist mit 90m die Flughöhe von Mission 3. Dabei handelt es sich um den Bereich der ehemaligen Terminals und Betriebsgebäude. Dort sind die größten Aktivitäten zu verzeichnen und gleichzeitig ist dies der Bereich, welcher sich zukünftig am meisten verändern wird. Aus diesen Gründen wurde die Entscheidung getroffen, diesen Bereich detaillierter zu erfassen und zu archivieren.

Tabelle 1: Eckdaten der Mission und Flugparameter

Mission	Flugstrecke in m	vorauss. Flugzeit in min	Anzahl Wegpunkte	Anzahl Photos	Kartierfläche in m ²	Flughöhe	Frontale Überlap-pung	Seitliche überlap-pung
1	17498	31	51	1119	568133	120	80	70
2	24887	44	46	1513	858289	120	80	70
3	38377	67	65	3107	1027463	90	80	70
4	52059	90	54	3163	1753642	120	80	70
Gesamt	132821	232	216	8902	4207527			

Alle vier Missionen wurden mit einer Fluggeschwindigkeit von 10m/s durchgeführt.



Abbildung 2-16: Screenshot des Flugmusters von Mission 1

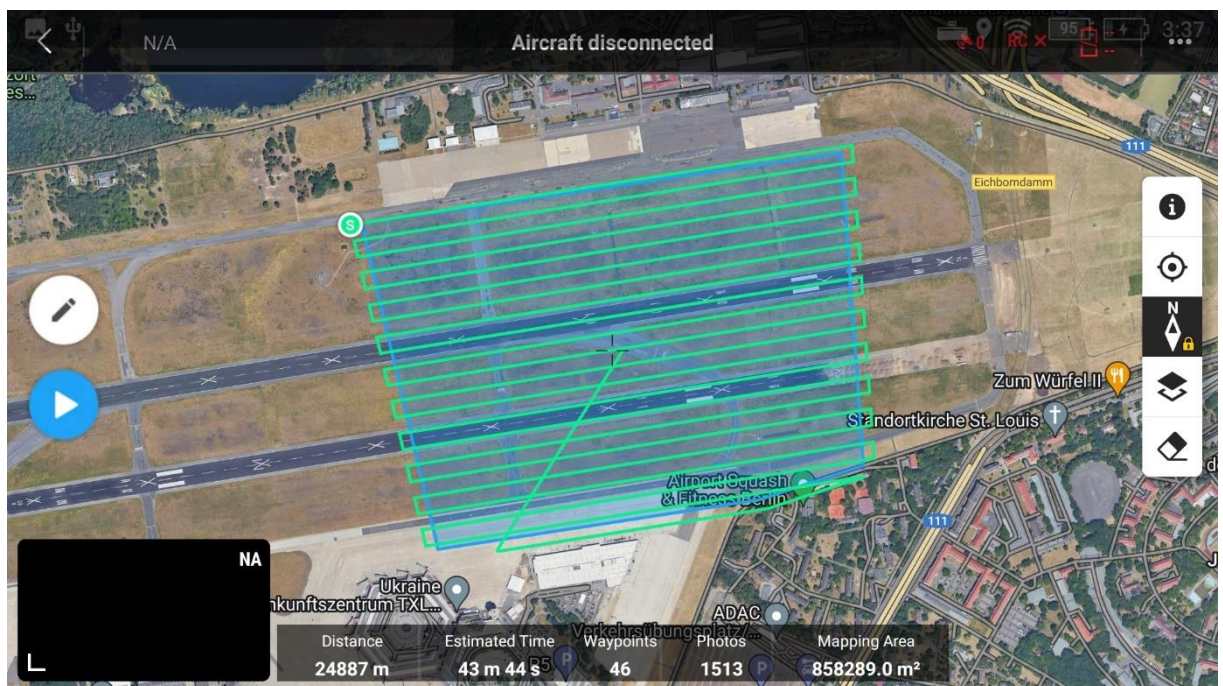


Abbildung 2-17: Screenshot des Flugmusters von Mission 2

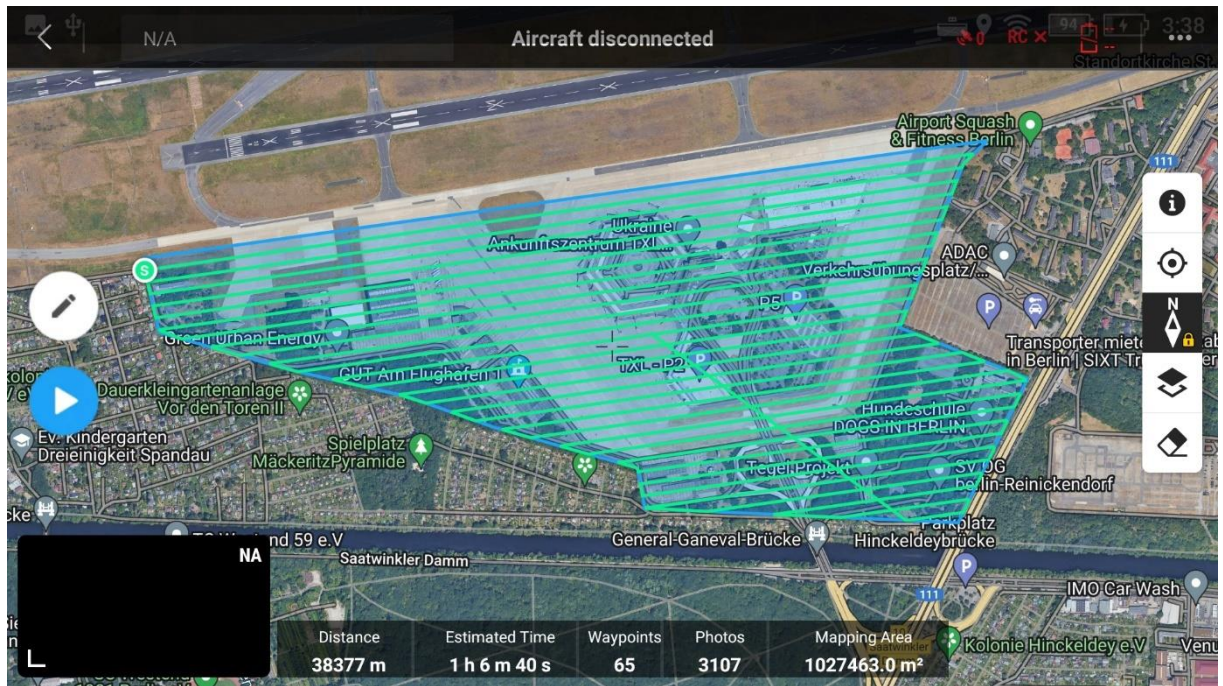


Abbildung 2-18: Screenshot des Flugmusters von Mission 3



Abbildung 2-19: Screenshot des Flugmusters von Mission 4

2.4 Automatisieren des Auswerteprozesses

Nachdem die Bildbefliegung abgeschlossen wurde, werden die relevanten Dateien und Ordner der beiden SD-Karten in einem neuen Ordner, welcher fortan als Projektordner bezeichnet wird, auf dem Prozessierungsrechner abgelegt. Der Projektordner wird immer nach dem gleichen Prinzip benannt, welches den Tag der Aufnahme enthält und sich wie folgt darstellt:

YYYYMMTT_P1_Gesamtareal

Anschließend werden eine BAT-Datei namens „start.bat“ sowie alle erforderlichen Python-Scripte aus einem Vorlagenorder ebenfalls in den Projektordner kopiert. Eine BAT-Datei (Batch-Datei) ist eine Textdatei, die eine Liste von Befehlen in der Windows-Eingabeaufforderung (cmd.exe) enthält. Die BAT-Datei kann als Skript ausgeführt werden, um mehrere Befehle in einer einzigen Anweisung auszuführen. BAT-Dateien werden häufig für die Automatisierung wiederkehrender Aufgaben und die Konfiguration von Systemeinstellungen wie dem Starten von Diensten und Scripten verwendet (IONOS-SE 2022). Die hier verwendete BAT-Datei wird verwendet, um die Software *Agisoft Metashape* in der Windows-Eingabeaufforderung zu starten und gibt einen Parameter mit, welcher das erste Python-Script mit der Bezeichnung *Main.py*³¹ in *Metashape* ausführt.

Sämtliche Scripte sowie die Bat-Datei befinden sich im Anhang sowie in diesem Gitlab-Repository:

<https://gitlab.com/unigis2/Masterthesis>

³¹ <https://gitlab.com/unigis2/Masterthesis/-/blob/main/Main.py>

2.4.1 Prozessübersicht

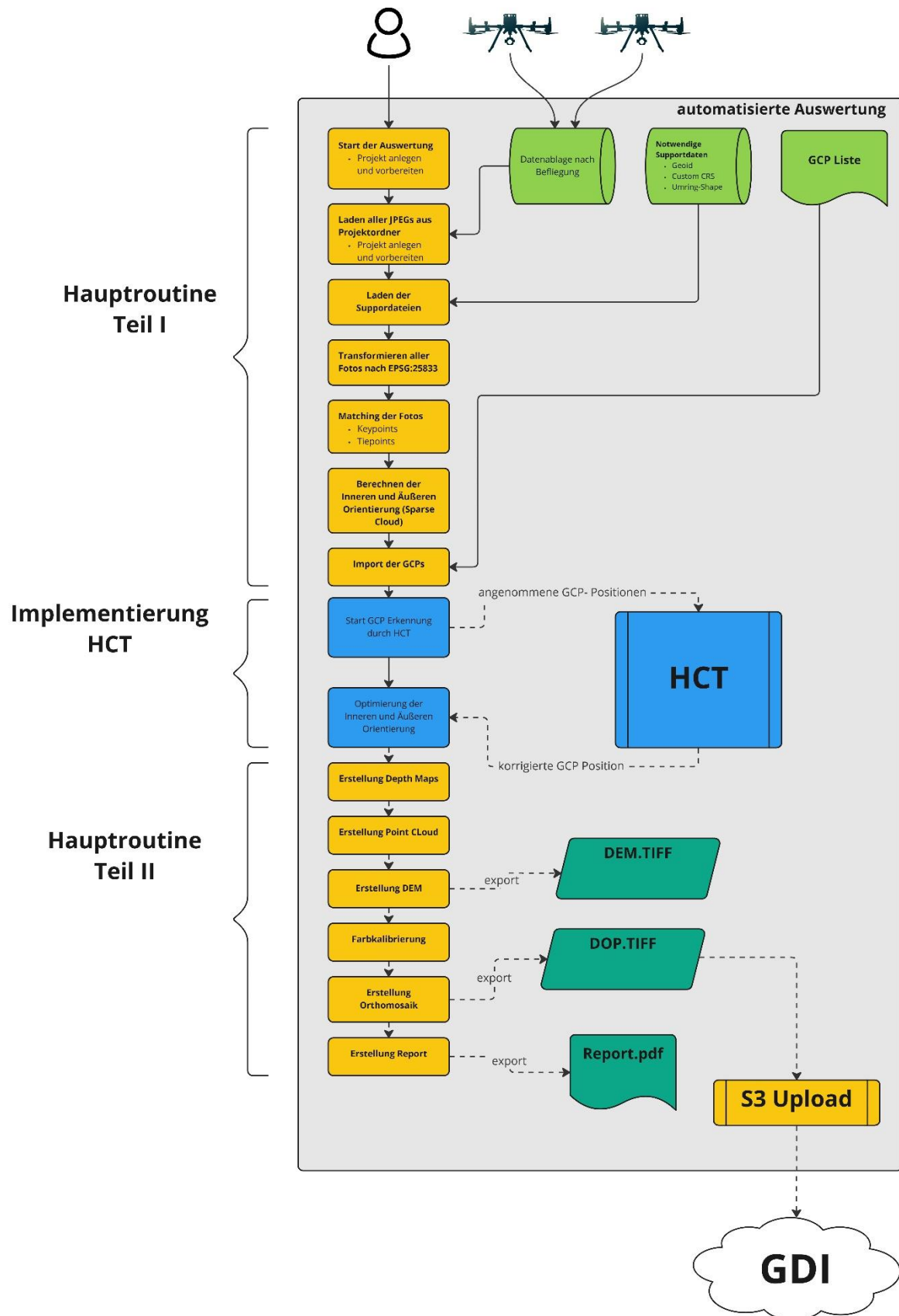


Abbildung 2-20: vereinfachte Darstellung des gesamten Auswertungsprozesses

2.4.2 Die Hauptroutine Teil I

Das Script *Main.py* startet den Auswerteprozess und beinhaltet die Verkettung aller erforderlichen Metashape-Funktionen, welche nötig sind, um aus den Rohdaten am Ende das DEM sowie das Orthomosaik zu erzeugen und in den S3-Speicher zu laden.

Zu Beginn werden alle erforderlichen Module in den Python-Interpreter importiert. Eine vollständige Aufstellung der verwendeten Module wurde im Abschnitt 2.2.4.4 gegeben.

Danach wird der Ordnerpfad indem sich das Script befindet, also der Pfad des Projektordners, in einer Variablen mit der Bezeichnung **homepath** gespeichert. Diese Variable wird im Gesamtablauf häufiger vorkommen, wenn es zur Verarbeitung physisch abgespeicherter Dateien kommt oder Datenmanagement durchgeführt wird.

Eine weitere Variable namens **tmppath** wird gespeichert. Diese enthält den Pfad eines Unterordners des Projektordners mit der Bezeichnung *tmp*. In diesem Ordner werden alle temporären Dateien, die während des Workflows entstehen, zwischengespeichert. Mit einer IF-Abfrage wird zunächst geprüft, ob der *tmp*-Ordner existiert. Falls ja, wird dieser inklusive aller enthaltenen Dateien und Unterordner gelöscht. Diese Abfrage wurde im Zuge der Testdurchläufe implementiert, um ständiges manuelles Löschen der Zwischenergebnisse zu vermeiden.

Anschließend wird ein neues Metashape-Projekt im Zwischenspeicher angelegt und in einer Variablen namens **doc** gespeichert. Über die Funktion **datetime** wird eine String-Variable erzeugt, die das aktuelle Datum als Grundlage für die Dateibezeichnung des Metashape-Projektes nach dem Schema *YYYYMMTT_Gesamtreal.psx* enthält, welches nun im Projektordner abgespeichert wird.

Nun wird dem Projekt durch **doc.addChunk()** ein Chunk hinzugefügt und in einer Variablen mit der Bezeichnung **chunk** gespeichert. Bei Chunks handelt es sich um logische Einheiten bzw. Gruppen, welche in Metashape zur Auf-

splitting von Projekten in Untereinheiten dienen. Sämtliche Prozesse können getrennt prozessiert und ggf. wieder zusammengefügt (merged) werden. Diese Aufteilung kommt beispielsweise in sehr großen Projekten zum Einsatz, bei denen die Kapazitäten des Computers nicht ausreichen würden um alle Daten in einem Schritt zu prozessieren. Die Prozessierung der einzelnen Chunks kann zudem auf mehrere Endgeräte verteilt werden. Im Falle der hier Entwickelten Anwendung wird lediglich ein Chunk benötigt. Die Ausstattung des eingesetzten Prozessierungsrechners wurde so gewählt, dass eine Aufteilung in Chunks nicht notwendig ist. (vgl. 2.2.3)

Nun erfolgt eine Umstellung des im Chunk definierten lokalen Koordinatensystems zum geodätischen Koordinatensystem *ETRS89* mit dem EPSG-Code 4258³². Die Koordinaten der in den Folgeschritten zu importierenden Fotos liegen durch die SAPOS-HEPS Korrektur (vgl. 2.2.4.3) in diesem Referenzsystem vor. Metashape liest beim Import der Fotos deren Metadaten (XMP, siehe 2.2.4.5) aus und interpretiert die darin enthaltenen Koordinaten als WGS84-Koordinaten (EPSG-Code 4326³³) und weist dieses System automatisch dem Chunk zu. Dabei handelt es sich um einen Fehler in den XMP-Metadaten, denn dort ist fälschlicherweise WGS84 deklariert. Um diesen Effekt zu verhindern, wird dem Chunk vorher das korrekte *ETRS89* zugewiesen. Die Koordinaten des Lagebezugs sind bei beiden genannten Systemen identisch. Ein minimaler Unterschied besteht in den Höhenkoordinaten, da beide Systeme jeweils ein anderes Referenzellipsoid verwenden. Im Bereich des Forschungsgebietes sind die gemessenen Höhen im *ETRS89* einheitlich 0.000066 Meter höher als gemessene Höhen im *WGS84*.

Nachfolgend geht es an den Import der aufgenommenen Fotos in das Projekt. Dafür wird zunächst eine leere Liste mit der Bezeichnung *camera_list* angelegt. Anschließend wird mit Hilfe der Methode `walk.os(homepath)` im Projektverzeichnis nach Dateien mit der Endung ".JPG" gesucht. Dabei wird durch jeden Ordner und jede Datei im Verzeichnisbaum mittels zwei for-

³² <https://epsg.io/4258>

³³ <https://epsg.io/4326>

Schleifen iteriert: Eine Schleife, um jeden Ordner und seine Unterverzeichnisse zu durchlaufen und eine weitere, um jede Datei im aktuellen Ordner zu überprüfen. Wenn eine Datei mit der Endung ".JPG" gefunden wurde, wird ihr vollständiger Pfad mithilfe von `os.path.join(root, filename)` ermittelt und anschließend mit `camera_list.append(os.path.join(root, filename))` zur vorher erstellten Liste namens `camera_list` hinzugefügt. Diese Schleifen und Überprüfungen werden für jede Datei und jeden Ordner im Verzeichnisbaum durchgeführt, bis alle Dateien mit der Endung ".JPG" gefunden und zur Liste `camera_list` hinzugefügt wurden. Mittels `chunk.addPhotos(camera_list)` werden alle Fotos aus der Liste in das Metashape-Projekt importiert. Durch diesen Prozessablauf ist es wichtig, dass die unter **Fehler! Verweisquelle konnte nicht gefunden werden.** beschriebene Datenauswertung sorgfältig durchgeführt wird, um den Import ungewollt erstellter Fotos zu verhindern. Darüber hinaus werden Funktionen die benötigten Parameter mitgegeben, um zusätzlich die relevanten XMP-Metadaten (vgl. 2.2.4.5) der Fotos zu importieren. Dadurch erhält die Software zu jedem Foto die RTK-korrigierten Koordinaten zur Lage und Höhe, die dazugehörigen Genauigkeitsangaben, Angaben der Achsneigung des 3-Achsen Gimbals zum Aufnahmezeitpunkt sowie Informationen zur inneren Orientierung der Kamera. Diese Werte wurden während des Fluges mithilfe der IMU sowie den RTK-Daten von der Drohnensoftware berechnet und fließen in Metashape in die Bündelblockausgleichsberechnungen ein, was eine signifikante Steigerung der Genauigkeit bzw. Qualität der Berechnungen bewirkt. (Przybilla, Reuber et al. 2015)

In den nachfolgenden Schritten wird ein benutzerdefiniertes Koordinatensystem (Custom CRS) für die Kameras im Chunk und für den Chunk selbst geladen und angewandt (vgl. 2.2.4.1.3). Es wird eine benutzerdefinierte CRS-Datei mit dem Namen `25833DHHN2016.prj` geöffnet, die im Verzeichnis `homepath` gespeichert ist. Dies geschieht mithilfe des `open`-Befehls, bei dem das Zielverzeichnis und der Dateiname übergeben werden. Die Datei wird im Lesemodus geöffnet, so dass nur gelesen, aber nicht geschrieben

oder verändert werden kann. Die erste Zeile der Datei, welche die benutzerdefinierten CRS-Informationen enthält, wird als Variable `custom_crs` gespeichert.

Danach wird die entsprechende Geoid-Datei (vgl. 2.2.4.2) geladen, welche im Metashape-Installationsverzeichnis gespeichert und in der Metashape-Registrierung mit dem Custom CRS verbunden ist, indem die `addGeoid`-Methode aufgerufen wird und die Geoid-Datei als Argument übergeben wird. Anschließend wird das Zielkoordinatensystem als `target_crs` definiert, indem ein `Metashape.CoordinateSystem-Objekt` erstellt und mit der Variablen `custom_crs` als Argument übergeben wird. Dann werden alle Kameras im Chunk mit einer `for`-Schleife durchlaufen. Für jede Kamera wird geprüft, ob ihre Referenzlage definiert ist, also ob sie über einen gültigen Lagebezug verfügt. Falls das nicht der Fall ist, wird die Schleife übersprungen. Im Normalfall sollte dies nicht vorkommen und dem Autor ist noch kein Fall diesbezüglich bekannt. Sollte der Fall jedoch eintreten, so verhindert diese Prüfung einen Abbruch der Routine. Für alle Kameras mit bekannter Referenzlage wird diese mittels der `transform`-Methode des `Metashape.CoordinateSystem-Objekts` von dem ursprünglichen Koordinatensystem (ETRS89) in das Zielkoordinatensystem (Lagebezug: ETRS 89 UTM-Zone 33; Höhenbezug DHHN2016) transformiert. Schließlich wird auch das aktuelle CRS des Chunks auf dieses Zielkoordinatensystem gesetzt, indem die `target_crs`-Variable als neues CRS für den Chunk definiert wird.

Anschließend werden die Änderungen im Metashape-Projekt gespeichert und per `print()` eine Statusmeldung für den Anwender ausgegeben. Das Speichern wird im weiteren Verlauf immer wieder stattfinden und wird zu Gunsten der Übersichtlichkeit nicht wieder erwähnt werden. Sämtliche Speicherpunkte können direkt in den Scripten im Anhang (vgl. 7) gesichtet werden.

Nun geht es an die eigentliche Durchführung der photogrammetrischen Berechnungen. Dieser Prozess nimmt ca. 10 Stunden Rechenzeit in Anspruch.

Mit den Funktionen `matchPhotos()` sowie `alignCameras()` werden Aerotriangulation und Bündelblockausgleich angestoßen sowie die Kamerakalibrierung optimiert. Beiden Funktionen werden diverse Parameter mitgegeben, deren Ausprägung sich nach zahlreichen Befliegungen und Auswertungen als optimal für dieses Projekt herausgestellt hat. Die Erörterung dieser Parameter und deren Funktion sind im Kontext untergeordnet relevant und können im Benutzerhandbuch der Software³⁴ sowie in der Dokumentation³⁵ der API recherchiert werden. Die für diese Arbeit gewählten spezifischen Werte sind entsprechend den Python-Codes im Anhang (vgl. 7) zu entnehmen.

Nach Abschluss dieser Berechnungen werden die Positionen der GCP's mittels `importReference()` aus der Datei *GCP.txt* in das Projekt geladen. Was danach folgt, ist die Korrektur der berechneten GCP-Positionen auf allen Fotos, welche einen GCP enthalten. In Metashape werden die GCP-Positionen als *Marker* bezeichnet. Diese Bezeichnung wird auch in den weiteren Ausführungen mehrfach verwendet und meint die GCPs.

³⁴ https://www.agisoft.com/pdf/metashape-pro_2_0_en.pdf

³⁵ https://www.agisoft.com/pdf/metashape_python_api_2_0_0.pdf

2.4.3 Implementierung der Hough Circle Transform Methode

2.4.3.1 Prozessübersicht

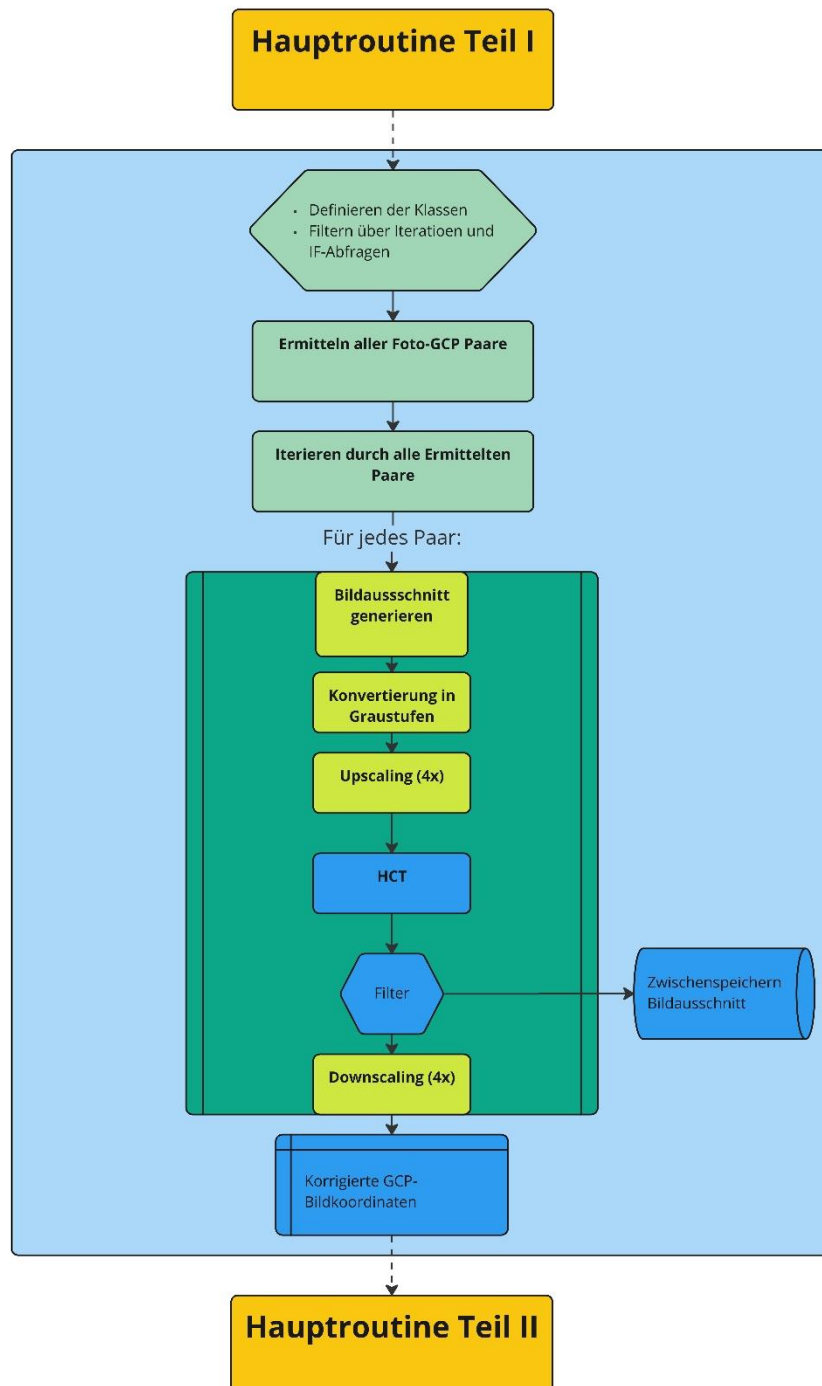


Abbildung 2-21: vereinfachte Darstellung des Ablaufs der implementierten HCT-Methode

2.4.3.2 Beschreibung

Dieser Schritt wurde bisher manuell mittels Mausklicks durchgeführt und wird nun durch die auf der *Hough Circle Transformation (HCT)* basierenden Methode ersetzt. In der folgenden Abbildung wird der Vergleich vor und nach der Korrektur mittels *HCT* veranschaulicht.



Abbildung 2-22: Marker vor der Korrektur (links) und Marker nach der Korrektur (rechts)

Der Prozess wurde in ein separates Modul mit der Bezeichnung *Set_Markers_HC.py* ausgelagert und wird aus der Hauptroutine (*Main.py*) mit der Syntax `mkset.set_markers(homepath, chunk, doc)` aufgerufen.

In diesem Modul werden zunächst alle weiteren, zur Ausführung notwendigen, Module geladen. Anschließend werden zwei verschiedene Klassen mit den benötigten Attributen definiert. Die Klasse `CMarkers()` wird später alle Marker-Kamera Kombinationen mit den jeweiligen Marker- und Kamerabezeichnungen sowie den Bildkoordinaten des Markers vor und nach der Korrektur durch *HCT* enthalten. Die Klasse `CVmatches()` wird verwendet, um mehrere durch *HCT* erhaltene Matches zu filtern. Die Funktion beider Klassen wird an den erforderlichen Abschnitten noch ausführlicher erläutert.

Die aus der Hauptroutine aufgerufene Funktion `set_markers()` übernimmt das Auslesen und Manipulieren der Kameras und Marker in Metashape. Zuerst wird eine leere Liste namens `marker_list` angelegt. Diese Liste wird mit allen gültigen Kamera-Marker Paaren, also allen Objekten der Klasse `CMarkers()` befüllt werden. Dazu wird über zwei `for`-Schleifen durch alle Marker und durch alle Kameras im Chunk iteriert. Anschließend werden verschiedene Prüfungen bzw. `if`-Abfragen durchgeführt.

Die erste Abfrage prüft, ob die Kamera über keine Verortung verfügt. Falls keine Verortung vorliegt, dann wird diese Kamera übersprungen. Dieser Fall könnte bei technischen Fehlfunktionen während der Befliegung oder Fehlern im Datenmanagement zustande kommen. Die Abfrage verhindert den daraus resultierenden Abbruch der Routine.

Die Abfrage `if camera.project(marker.position):` überprüft, ob eine Kamera eine bestimmte Markierung in ihrem Sichtfeld hat. Mit der Methode `camera.project(marker.position)` wird berechnet, ob die Projektion eines Markers (`marker.position`) in das zweidimensionale Bild einer Kamera fällt. Wenn das Projektionsergebnis gültig ist (nicht None), werden die Koordinaten des Markers in die Variablen `cmposx` und `cmposy` geschrieben. Auch der Pitch-Wert der Kamera (vgl. 2.2.1.1) wird in eine Variable (`pitch`) geschrieben und anschließend mit den Schwellenwerten $+1^\circ$ bzw. -1° gefiltert. Dadurch ist gewährleistet, dass nur Nadir-Aufnahmen für die weiteren Schritte verwendet werden. Die Schwellenwerte sind großzügig gewählt und könnten ggf. auch weiter eingegrenzt werden.

Als nächstes erfolgt eine Abfrage, die überprüft, ob die Koordinaten des Markers in die Ausdehnung des Kamerasensors fallen. Falls dem so ist, handelt es sich definitiv um ein Marker-Kamera Paar. Es wird ein Objekt der Klasse `CVmatches()` gebildet und mit den vier bekannten Attributen (Markerbezeichnung, Kamerabezeichnung, X-Koordinate des Markers, Y-Koordinate des Markers) der eingangs deklarierten Liste `marker_list` hinzugefügt.

Wurden alle Kameras und Marker durchlaufen, wird durch jeden Eintrag der gefüllten `marker_list` iteriert und die Funktion `opencv()` mit dem Listenobjekt (Objekt der Klasse `CVmatches()`) als Eingabeparameter aufgerufen. Jedes Objekt der `marker_list` steht für ein Foto, auf welchem ein GCP verortet ist. Innerhalb der aufgerufenen Funktion `opencv()` wird jedes dieser Fotos mittels Hough Circle Transform nach dem Kreis durchsucht, welcher dem GCP entspricht und dessen Mittelpunkt als korrigierte Bildkoordinaten in die

Photogrammetrie-Software zurückgegeben. Der exakte Verfahrensablauf innerhalb der Funktion stellt sich wie folgt dar:

Da sämtliche Schritte innerhalb der Funktion hauptsächlich mit der freien Bibliothek *openCV* unabhängig von Metashape durchgeführt werden, muss das zu durchsuchende Foto geladen werden. Dafür wird zunächst mithilfe der Funktion `search_jpg()` im Projektverzeichnis rekursiv nach der Datei über den Kameranamen (`obj._cname`) aus Metashape gesucht und in der Variablen `filename` gespeichert. Um Abbrüche des Scripts zu vermeiden, wird vorher über eine if-Abfrage geprüft, ob es überhaupt einen Kameranamen gibt. Anschließend wird das Foto über `cv.imread()` geladen und in der Variablen `src` gespeichert. Es erfolgt eine weitere if-Abfrage, ob das Foto fehlerfrei geladen werden konnte. Falls nicht, wird eine Print-Meldung an den Nutzer ausgegeben und die Funktion ist beendet.

Wurde das Foto wie erwartet geladen, geht es mit den nächsten Schritten weiter. Durch die RTK-Korrektur und die vorher durchgeführte Prozessierung in Metashape beträgt die Abweichung zwischen der projizierten GCP-Koordinate und dem GCP-Mittelpunkt auf dem Foto unter normalen Umständen nur wenige Zentimeter ($\leq 10\text{cm}$). Das bedeutet, dass nicht das vollständige Foto nach dem GCP-Kreis durchsucht werden muss und die Überprüfung eines reduzierten Bildausschnittes genügt. Daraus ergeben sich mehrere Vorteile. Es müssen geringere Datenmengen verarbeitet werden, was wiederum eine kürzere Laufzeit und damit gleichzeitig einen reduzierten Energieverbrauch zur Folge hat. Zudem bietet die Eingrenzung des zu durchsuchenden Bildausschnittes bei der Kreiserkennung durch *HCT* weniger Möglichkeiten für false Positives – Treffer, die keine sind.

Um den reduzierten Bildausschnitt aus dem Foto zu extrahieren, wird eine Bounding Box (BB) über XY-Koordinaten (`bbX1`, `bbY1`, `bbX2`, `bbY2`) mit einem Offset von -75 bzw. +76 Pixeln ausgehend von der gerundeten Bildkoordinate des nicht korrigierten Markers definiert. Bei einer GSD von 1,13cm (bei 90m Flughöhe) ergibt sich dadurch ein Bildausschnitt mit einer Flächenabdeckung von $\sim 0,85\text{m} \times 0,85\text{m}$. Bei einer angenommenen Abweichung zwischen gemessener und korrigierter Koordinate $\leq 10\text{ cm}$ ist die

Größe des Bildausschnittes großzügig gewählt und lässt auch Korrekturberechnungen mit größeren Abweichungen (z.B. Fotos aufgenommen ohne RTK-fix) zu.

Anschließend werden die Mittelpunkt-Koordinaten (mX , mY) des Bildausschnittes (= Markerposition vor der Korrektur) für spätere Berechnungen gespeichert.

Bedingt durch die automatische Befliegung kann es mitunter vorkommen, dass ein GCP so dicht am Bildrand verortet ist, dass die definierte Bounding Box partiell in mindestens einer Richtung außerhalb des Bildbereiches fällt. Um einen Abbruch der Routine zu umgehen, wird für alle 4 BB-Koordinaten über if-Abfragen ein Abgleich mit der Ausdehnung des Bildes durchgeführt und ggf. eine Korrektur der Koordinaten und damit eine Reduzierung der BB gerechnet. Sollte es Korrekturen der BB-Corner im oberen linken Bereich geben, so werden entsprechend auch die Werte von mX bzw. mY angepasst. Dies ist notwendig, weil die Verortung dieser Koordinaten sich auf BB-Koordinaten $x = 0$ und $y = 0$ beziehen.

Anschließend wird der Bildausschnitt über die BB-Koordinaten abgegriffen und in der variablen `markerarea` gespeichert. Da die HTC-Methode als Eingabeparameter ein Graustufenbild benötigt (vgl. 2.1.3), wird der Bildausschnitt mit der `openCV`-Funktion `cvtColor(markerarea, cv.COLOR_BGR2GRAY)` in ein Graustufenbild konvertiert.

Bevor es an die Anwendung von *HCT* geht, wird der Bildausschnitt durch die Funktion `pyrUp()` um das 4fache hochskaliert (Upscaling). Dieser Schritt wurde implementiert, weil die durch *HCT* berechneten Mittelpunktkoordinaten eines detektierten Kreises lediglich in halben bzw. ganzen Pixelwerten zurückgegeben werden (z.B. $x = 230$; $y = 185.5$). Die tatsächliche Kreismitte lässt sich so jedoch nur in den seltensten Fällen exakt bestimmen. Durch das 4fache Upscaling wird die Präzision der Mittelpunktkoordinaten somit erhöht (siehe Abbildung 2-23). Die ermittelten Koordinaten werden anschließend auf das Niveau vor dem Upscaling zurückgerechnet (z.B. $x = 345.5 * 0,25 = \mathbf{86,375}$; $y = 281 * 0,25 = \mathbf{70,25}$)

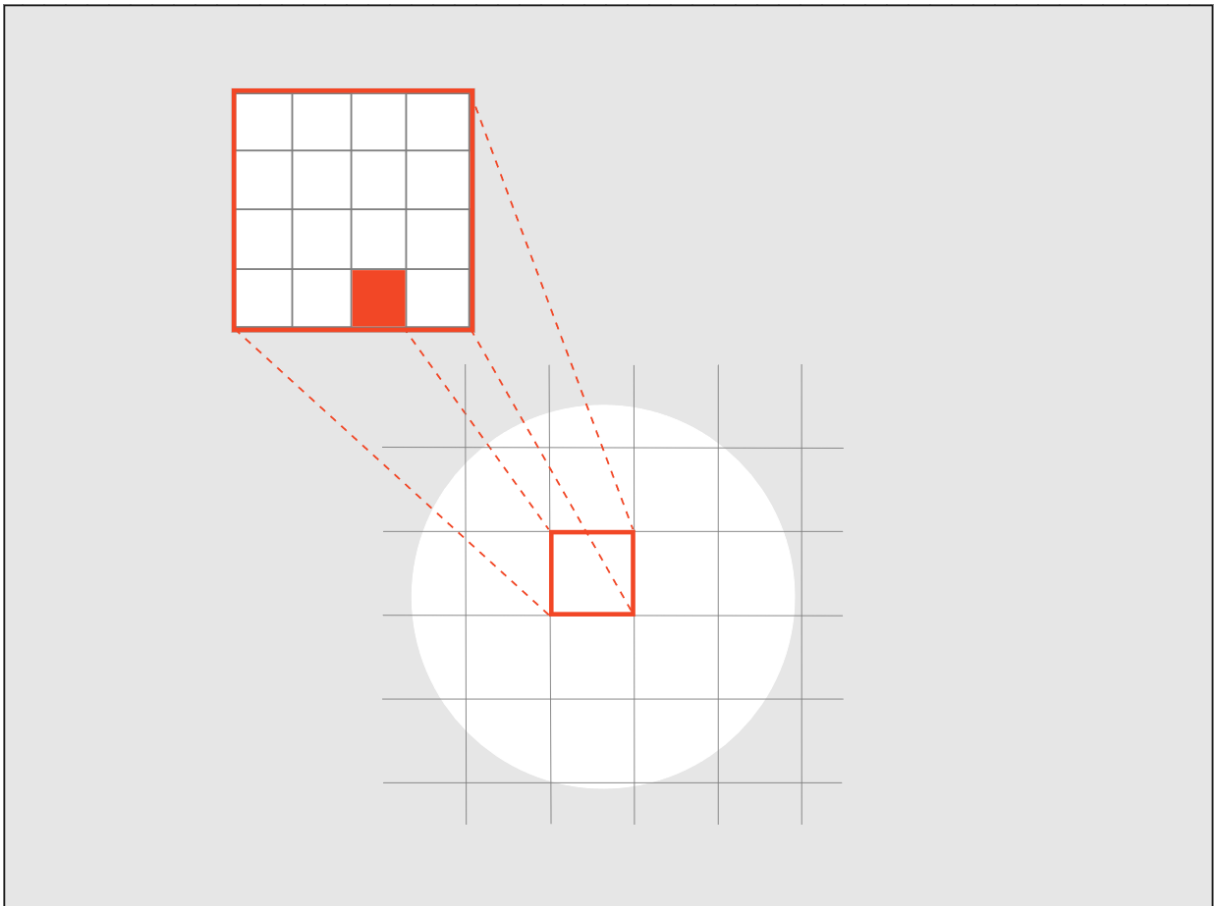


Abbildung 2-23: vereinfachte Darstellung des Upscalings

Schließlich wird der hochskalierte Bildausschnitt mit der Funktion **HoughCircles()** per *HTC* nach Kreisen durchsucht. Die Parametrierung dieser Funktion spielt dabei eine wichtige Rolle, denn ungünstige Parameter führen zu keinen bzw. falschen Ergebnissen. Zum besseren Verständnis der hier gewählten Werte wird im Folgenden näher auf die Bedeutung dieser Parameter eingegangen. Sämtliche Erklärungen sind grundsätzlich in der offiziellen Dokumentation von *openCV* zu finden (OpenCV 2022)

`HoughCircles(image, method, dp, minDist, param1, param2, minRadius, maxRadius)`

- **image:**
 - Das Eingaberaster (hier: `gray`)
- **Method:**
 - Die verfügbaren Erkennungsmethoden sind `HOUGH_GRADIENT` und `HOUGH_GRADIENT_ALT`. Im Rahmen dieser Arbeit wurde die Methode `HOUGH_GRADIENT` verwendet. Es wurden beide Methoden mit diversen Parametrierungen getestet und es konnte kein Unterschied festgestellt werden.
- **dp:**
 - umgekehrter Skalierungsfaktor für die Bildverarbeitung. Der hier gewählte Wert `1` bedeutet, dass keine Skalierung durchgeführt wird.
- **minDist:**
 - Der Mindestabstand zwischen den Zentren der erkannten Kreise. Ist der Parameter zu klein, werden fälschlicherweise mehrere Nachbarkreise zusätzlich zu einem richtigen Kreis erkannt. Ist er zu groß, können einige Kreise übersehen werden. Im Rahmen dieser Arbeit wurde der Wert auf `1000` Pixel festgelegt. Bei einer Seitenlänge von $604 * 604$ Pixeln (oder weniger bei Reduzierung durch Randlagen) würde die gemeinsame Hypotenuse der beiden teilenden rechtwinkligen Dreiecke `854,185` Pixel betragen. Soll nur ein Kreis auf dem Bild gefunden werden, dann muss der Wert des Parameters größer als der errechnete Wert sein.

- **param1:**
 - Dies ist der erste methodenspezifische Parameter bestimmt die Empfindlichkeit der Kanten, d.h. wie stark die Kanten des Kreises sein müssen, um erkannt zu werden. Wenn der Wert zu hoch ist, werden keine Kanten erkannt, während ein zu niedriger Wert zu viel Rauschen erzeugt. Im Rahmen dieser Arbeit wurde der Wert auf **50** gesetzt. Ermittelt wurde der Wert durch zahlreiche Testläufe.

- **param2:**
 - Dies ist der zweite methodenspezifische Parameter. Er legt fest, wie viele Kantenpunkte erforderlich sind, um einen Kreis zu erkennen. Ein zu hoher Wert führt dazu, dass keine Kreise erkannt werden, während ein zu niedriger Wert dazu führt, dass zu viele Kanten als Kreis erkannt werden. Der optimale Wert von Parameter 2 hängt von der Größe der Kreise ab. Im Rahmen dieser Arbeit wurde der Wert auf **30** gesetzt. Ermittelt wurde der Wert durch zahlreiche Testläufe.

- **minRadius:**
 - Der kleinste Radius, den ein gefundener Kreis haben darf. Im Rahmen dieser Arbeit wurde der Wert auf **35** gesetzt. Der Wert leitet sich aus der GSD von 1,51 cm auf 120m Flughöhe bei einem Kreisradius von 15 cm und unter Berücksichtigung des 4-fachen Upscalings ab. ($\text{minRadius} = 15 / 1,51 * 4 = \sim 39,70$)
 - Der Parameter wurde um einen Toleranzwert von ca. 5 Pixeln verkleinert, um die unterschiedlichen Geländehöhen abzufedern.

- **maxRadius:**
 - Der größte Radius, den ein gefundener Kreis haben darf. Im Rahmen dieser Arbeit wurde der Wert auf **60** gesetzt. Der Wert leitet sich aus der GSD von 1,13 cm auf 90m Flughöhe bei einem Kreisradius von 15 cm und unter Berücksichtigung des 4-fachen Upscalings ab. ($\text{maxRadius} = 15 / 1,13 * 4 = \sim 53,10$)
 - Der Parameter wurde um einen Toleranzwert von ca. 7 Pixeln vergrößert, um die unterschiedlichen Geländehöhen abzufedern.

Wurde die Kreiserkennung durchgeführt erfolgt durch eine if-Abfrage die Prüfung, ob mindestens ein Kreis gefunden wurde. Falls kein Kreis gefunden wurde, ist die Funktion beendet. Andernfalls läuft die Funktion weiter und legt eine leere Liste mit der Bezeichnung **objdistlist** an. In dieser Liste wird über eine for-Schleife für jeden gefundenen Kreis im Bildausschnitt ein Objekt der Klasse **CVmatches** mit den dazugehörigen Mittelpunktkoordinaten (**Px**, **Py**), dem Radius (**rad**) sowie der mittels **math.dist()** berechneten Distanz (**eDistance**) zwischen dem Mittelpunkt des gefundenen Kreises und der Bildkoordinate der Markerposition vor Korrektur durch den Befehl **objdistlist.append()** gespeichert.

Anschließend wird die Liste nach dem Objekt mit der geringsten Distanz durchsucht und der Variablen **min_dist** zugewiesen. Um ggf. einen ungewollten Abbruch der Funktion zu verhindern, wird vorher mit einer if-Abfrage überprüft, ob die **objdistlist** mindestens einen Eintrag besitzt. Durch die in **HoughCircles()** gesetzten Parameter kann es maximal einen Eintrag geben. Die Kreiserkennung in der ursprünglichen Fassung dieser Funktion betrachtete das gesamte Bild statt des reduzierten Bildausschnittes. Trotz vieler Iterationsschleifen mit unterschiedlichen Varianten der Parametrierung konnte die Kreiserkennung durch andere kreisähnliche Strukturen wie beispielweise die ehemalige Flughafenbefeuerung, Schieberdeckel oder Entlüftungen etc. nicht ausschließlich auf die GCP's gelenkt werden. Ausgehend

von der hohen RTK-Genauigkeit und der gewählten Positionierung der GCP's im Felde konnte davon ausgegangen werden, dass der gefundene Kreis mit der geringsten Distanz zur Markerposition vor der Korrektur der richtige war. Aus den genannten Gründen ist die beschriebene Methode zur Ermittlung der kürzesten Distanz nicht mehr zwingend notwendig, wurde ungeachtet dessen jedoch beibehalten. Zum einen besteht nun immer noch die Möglichkeit, den Distanzabgleich für andere Szenarien zu verwenden. Dabei könnte es sich beispielsweise um eine Vergrößerung des Bildausschnittes oder eine erneute Betrachtung des Gesamtbildes handeln. Gegebenenfalls könnte es in einem Szenario auch mehr als einen GCP pro Bild geben. Zum anderen wird die Methode für den Abgleich des berechneten Distanzwertes mit einem definierten Schwellenwert von 200 Pixeln verwendet. Dies entspricht einer maximalen Entfernung von $1,13\text{cm} / 4 * 200 = 56,5\text{ cm}$. Alle erkannten Kreise mit einer größeren Distanz werden nicht berücksichtigt.

Ist die Distanz kleiner gleich als der Schwellenwert, wird die Korrekturkoordinate des Markers berechnet. In diesem Schritt werden die Koordinaten des gefunden Kreismittelpunktes durch den Upscaling-Faktor (hier 4) dividiert und damit in das Niveau des originalen Bildausschnittes gebracht. Anschließend werden die Koordinaten jeweils auf die Koordinaten der upper-left-Corner der BB addiert. Die beiden Korrekturkoordinaten (**obj._mprojcx**, **obj._mprojcy**) werden dann am Ende der Funktion mittels **return**-Befehl zurückgegeben.

Zur besseren Nachvollziehbarkeit werden der originale sowie der hochskalierte Bildausschnitt im *tmp*-Ordner gespeichert. In beiden Varianten wird die unkorrigierte Markerkoordinate mit einem roten Pixel dargestellt. Der gefundene Kreis wird mit einer grünen Umrandung sowie einem grünen Pixel für den Mittelpunkt dargestellt. Die gespeicherten Grafiken sind eindeutig benannt und enthalten den Dateinamen des Originalbildes sowie die Markernummer am Ende.

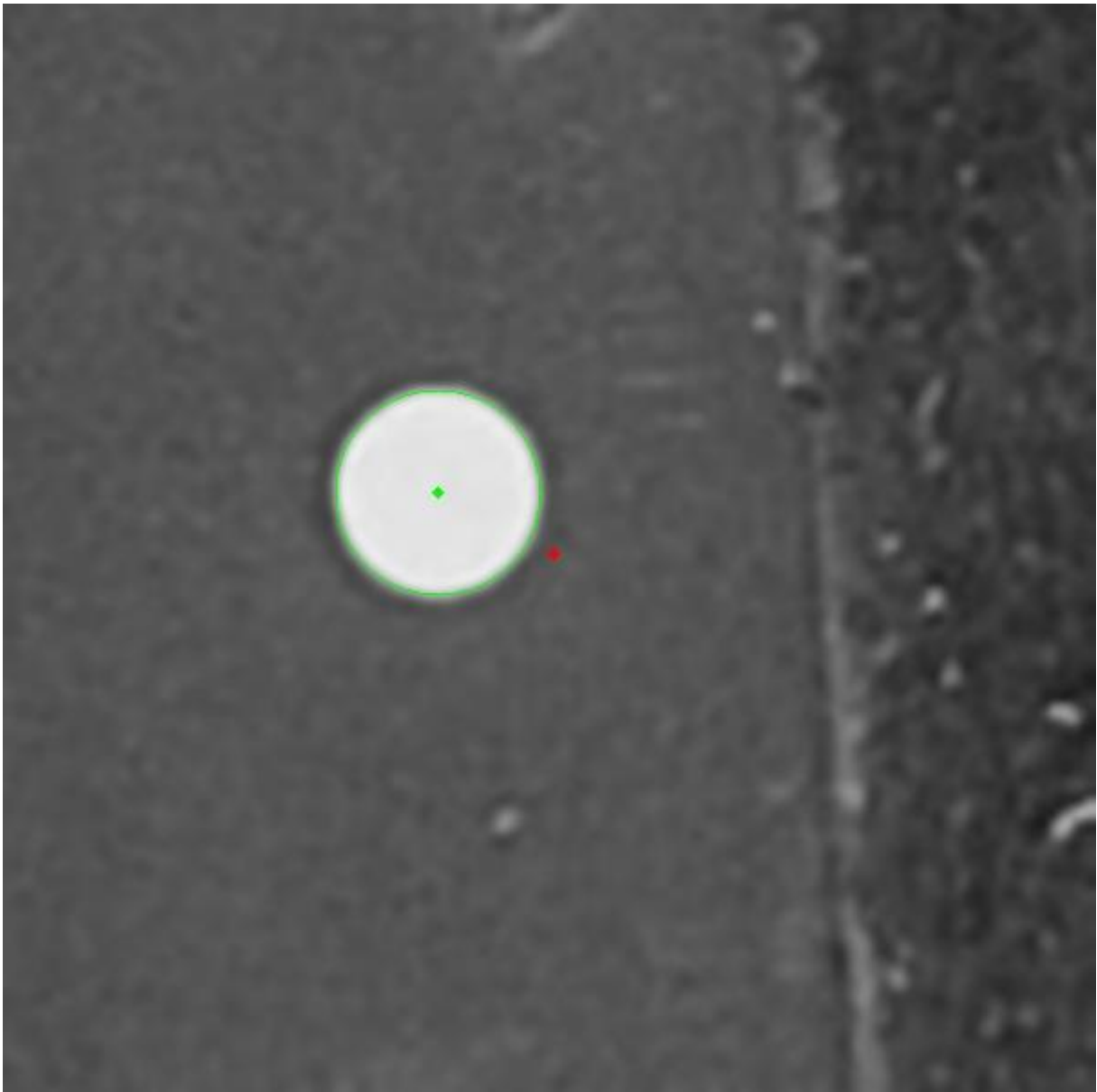


Abbildung 2-24: Erkannter GCP mit Mittelpunkt (grün) und Position vor Korrektur (rot)

Nachdem alle Kamera-Marker Paare durchsucht und die entsprechenden Korrekturkoordinaten berechnet wurden, müssen letztere als neue (bzw. korrigierte) Markerposition in Metashape deklariert werden. Dazu wird in der vom Mainscript (`Main.py`) aufgerufenen Funktion `set_markers()` über zwei for-Schleifen nochmals durch alle Marker und Kameras iteriert.

Anschließend werden wieder verschiedene Prüfungen durch if-Abfragen durchgeführt, um bei eventuellen Ausnahmen einen ungewollten Programmabsturz zu verhindern. Mit einer weiteren for-Schleife wird zusätzlich durch alle Objekte der `marker_list` iteriert. Danach erfolgt über `if`

`newdest._mlabel == marker.label and newdest._cname == camera.label` die Zuordnung der Listenobjekte zu den Kamera-Marker Kombinationen aus Metashape. Mit einer letzten if-Abfrage wird überprüft, ob entweder der x-Wert (`_mprojcx`) oder der y-Wert (`_mprojcy`) der Korrekturkoordinaten des Listenobjekts größer als der Default-Wert (0) ist. Falls dies zutrifft, wird die Marker-Projektion auf der Kamera durch die Korrekturwerte ersetzt.

Damit wurde die Korrektur der Marker mithilfe der Hough Circle Transformation durchgeführt und die Hauptroutine (`Main.py`) läuft an dieser Stelle weiter.

2.4.4 Die Hauptroutine Teil II

Bei allen weiteren folgenden Metashape-Funktionen in der Hauptroutine werden diverse, teils obligatorische Parameter mitgegeben, deren Ausprägung sich nach zahlreichen Befliegungen und Auswertungen als optimal für dieses Projekt herausgestellt hat. Falls die Bedeutung eines oder mehrerer Parameter für das Verständnis des Lesers oder den weiteren Ablauf relevant ist, wird darauf näher eingegangen. Alle weiteren für diese Arbeit gewählten spezifischen Werte sind entsprechend den Python-Codes im Anhang (vgl. 7) zu entnehmen.

Im nächsten Schritt der Hauptroutine werden mittels `optimizeCameras()`, auf Basis der zuvor durchgeführten Anpassungen der Markerprojektionen, die innere und äußere Orientierung der beiden eingesetzten Kameras sowie die triangulierten Verknüpfungspunkt-Koordinaten verfeinert (vgl. 2.1.2).

Nachdem die Optimierung abgeschlossen wurde, folgt die durch `buildDepthMaps()` ausgelöste Erstellung der Tiefenkarten. Diese werden von der Software als Grundlage zur Berechnung der im Anschluss durch `buildPointCloud()` erstellten, hochaufgelösten Punktwolke verwendet und in hoher Qualität generiert. Ein wichtiger Parameter ist `point_confidence = True`, welcher mit der Punktkonfidenz einen Vertrauenswert für jeden Punkt in der Punktwolke generiert. Mithilfe dieses Wertes werden anschließend durch `setConfidenceFilter(min_confidence = 3, max_confidence = 255)` alle Punkte mit einem Konfidenzwert größer gleich 3 für den nächsten Schritt

gefiltert, um schwache bzw. ungenaue Punkte in den weiteren Schritten nicht zu berücksichtigen.

Anschließend wird das digitale Höhenmodell (Digital Elevation Model = DEM) durch den Befehl `buildDem()` generiert. Das Höhenmodell ist die Basis bzw. die Projektionsfläche, auf der schließlich das Orthomosaik erstellt wird (`buildOrthomosaic()`).

Bevor die Endprodukte (DEM + Orthomosaik) als GeoTIFF für die Weiterverwendung exportiert werden können, wird über `chunk.importShapes()` ein spezifisches Boundary-Shape importiert. Dieses definiert die Außengrenze der Exporte und stellt im Wesentlichen den Umring der Liegenschaft dar. Sämtliche Bereiche außerhalb des Boundary-Shapes werden somit nicht exportiert.

Nun erfolgt der Export der beiden Endprodukte (jeweils über: `exportRaster()`) in den Projektordner. Es werden diverse formatspezifische Parameter mitgegeben, welche beispielsweise die Komprimierung der Dateien definieren und den Dateinamen festlegen. Der Aufbau der Dateibezeichnung des Orthomosaiks ist relevant für die Weitergabe an den S3-Speicher und die darauf zugreifende Routine, welche aus dem Orthomosaik vollautomatisiert einen WMTS-Dienst erzeugt und diesen als neuen Layer in das WebGIS integriert (vgl. 2.2.4.5). Der Upload in den S3-Speicher ist der letzte Schritt in der Hauptroutine und damit das Ende, der im Rahmen dieser Arbeit betrachteten Prozesskette. Ausgelöst wird der Upload durch Aufruf des Befehls `s3.upload()` zusammen mit dem kompletten Dateipfad des Orthomosaiks als Parameter.

2.4.5 Upload in den Cloudspeicher

Für den Upload des Orthomosaiks in den S3-Speicher kommt ebenfalls ein kurzes Python-Script mit der Bezeichnung *S3_Upload.py* zum Einsatz. Aus Sicherheitsgründen ist dieses Script kein Bestandteil des Gitlab-Repositories bzw. des Anhangs. Daher wird sich an dieser Stelle auf eine kurze Erläuterung der Funktionsweise beschränkt. Der Ablauf der Routine ist einfach gehalten und besteht grundlegend aus den folgenden vier Schritten;

Im ersten Schritt wird die Erweiterung *Boto3* importiert.

Im zweiten Schritt werden alle benötigten Zugangsdaten sowie Verbindungsparameter definiert. Im dritten Schritt wird die Verbindung zum Cloudspeicher aufgebaut. Im vierten und letzten Schritt erfolgt der Dateiupload.

3 Ergebnisse

In diesem Kapitel werden die Auswertungsergebnisse der vier durchgeführten Befliegungen vorgestellt. Sämtliche Auswertungen wurden ausschließlich durch die in Kapitel 2.4 vorgestellten Routinen durchgeführt. Lediglich die Referenzierung von Checkpoints wurde manuell durchgeführt. Die eigentlichen Produkte, also die DEMs und Orthomosaik jeder Befliegung können nur als Screenshot abgebildet werden. Aufgrund der Dateigrößen (je DEM ca. 20GB und je Orthomosaik ca. 70GB) können diese nicht als Anhang bzw. mittels Gitlab übergeben werden. Ein Zugang zum Geoportal kann aus Datenschutzgründen leider nicht pauschal eingerichtet werden.

Der Fokus der hier vorgestellten Ergebnisse liegt auf den Indikatoren zum Nachweis der Funktionalität der implementierten GCP-Erkennung mittels Hough Circle Transform sowie auf der Bestimmung der erreichten Positionsgenauigkeiten jeder Befliegung.

Nachweis der Funktionalität von HCT:

Die Funktionalität der implementierten Methode wurde in drei Schritten überprüft. Zuerst wurden alle erkannten GCPs manuell überprüft. Wie in den Erläuterungen in Kapitel 2.4.3 beschrieben, wird der analysierte Bildausschnitt für jeden erkannten GCP in einem Ordner im Projektverzeichnis gespeichert. Auf jedem Bildausschnitt ist die durch Metashape angenommene Position des GCPs vor der Korrektur (roter Punkt) und der nun erkannte GCP durch einen grünen Kreis sowie einen grünen Punkt dargestellt. Der grüne Punkt stellt den Mittelpunkt des Kreises dar und ist dementsprechend die Korrekturposition des GCPs. Die folgende Abbildung zeigt ein Beispiel zur Veranschaulichung:



Abbildung 3-1: Beispiel eines durch HCT erkannten GCPs. Die gelben Elemente wurden zur Veranschaulichung in Nachhinein hinzugefügt.

Sämtliche Bildausschnitte wurden für alle 4 Befliegungen auf Auffälligkeiten wie falsche Treffer oder unpräzise Treffer geprüft.

Im zweiten Schritt wurden die korrigierten Marker (GCPs) in Agisoft Metashape manuell überprüft. Das Augenmerk wurde dabei auf die korrekte Positionierung der Marker gelegt und es wurde auf Auffälligkeiten wie beispielsweise hohe Fehlerwerte geachtet.

Im dritten und letzten Schritt wurden mehrere Größen zur Ermittlung der Erkennungsraten bestimmt und tabellarisch sowie graphisch ausgewertet. Die betrachteten Kenngrößen lauten wie folgt:

- maximal mögliche erkennbare Anzahl von Fotos je GCP ermittelt.
- Tatsächliche Anzahl der Fotos auf denen GCPs erkannt wurden
- Tatsächliche Anzahl der Fotos auf denen GCPs nicht erkannt wurden
- Berechnete Erkennungsraten einzeln je GCP und je Befliegung

Bestimmung der Positionsgenauigkeit:

Die Positionsgenauigkeit wurde in erster Linie mithilfe von Checkpoints alphanumerisch überprüft.

Checkpoints werden in Metashape, genau wie die GCPs, als Marker angelegt und auf die gleiche Weise referenziert. Der Unterschied zu den GCPs besteht darin, dass Checkpoints nicht als Eingabeparameter in den Auswertelgorithmen berücksichtigt werden. Die Software berechnet stattdessen den RMSE (Root Mean Square Error) für die Lage- und Höhenkoordinaten der Checkpoints. Der RMSE gibt an, wie weit die angegebenen Koordinaten eines Checkpoints von den in Metashape berechneten Koordinaten, also dem Resultat der Befliegung, im Durchschnitt entfernt sind. Es handelt sich dabei um einen Soll-Ist-Vergleich. Die Fehlerwerte aller ausgewerteten Befliegungen wurden grafisch visualisiert und gegenübergestellt. Die Einheit aller Fehlerwerte beträgt Zentimeter (cm). Die Vermarkung der Checkpoints wurde mit Luftbildplatten (siehe Abbildung 3-2) realisiert und die Positionsbestimmung mithilfe des Leica FLX100 GNSS Messgerätes durchgeführt.

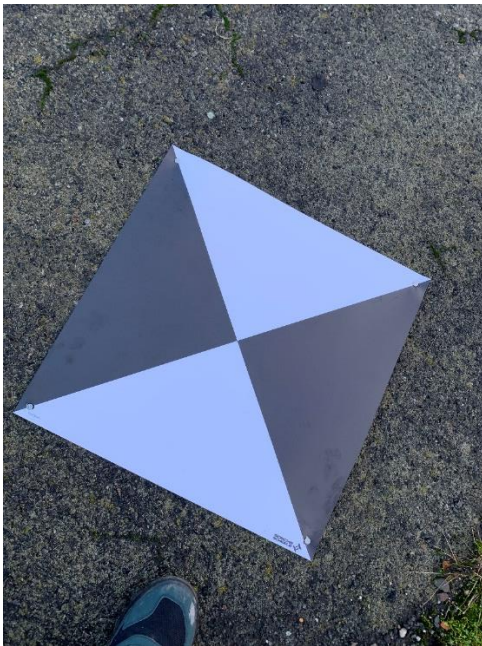


Abbildung 3-2: Luftbildplatte zur Vermarkung eines Checkpoints

Insgesamt wurden neun verschiedene Checkpoints im Forschungsgebiet angelegt. Die Verteilung wurde eher stichprobenartig als gleichmäßig verteilt gewählt, da die Qualität bzw. die Positionsgenauigkeit der erzeugten Daten an jeder Stelle gegeben sein muss. Die Verteilung der Checkpoints ist in der folgenden Abbildung dargestellt:



Abbildung 3-3: kartografische Darstellung der angelegten Checkpoints

Zu guter Letzt wurden subjektive Sichtvergleiche in ArcGIS Pro durchgeführt. Die relative Lagegenauigkeit der 4 erzeugten Luftbilder sowie älterer Aufnahmen wurde durch übereinanderlegen und mithilfe des SWIPE-Tools stichprobenartig auf Versprünge bzw. Lageabweichungen abgeglichen. Darüber hinaus wurde, ebenfalls in ArcGIS Pro, ein Sichtvergleich mit einem topographischen Bestandsplan aus Vermessungen durchgeführt. Das Fazit dieser beschriebenen Vergleiche wird in Kapitel 4 thematisiert.

3.1 Ergebnis Befliegung 1 (02.02.2023)

3.1.1 Orthomosaik und DEM

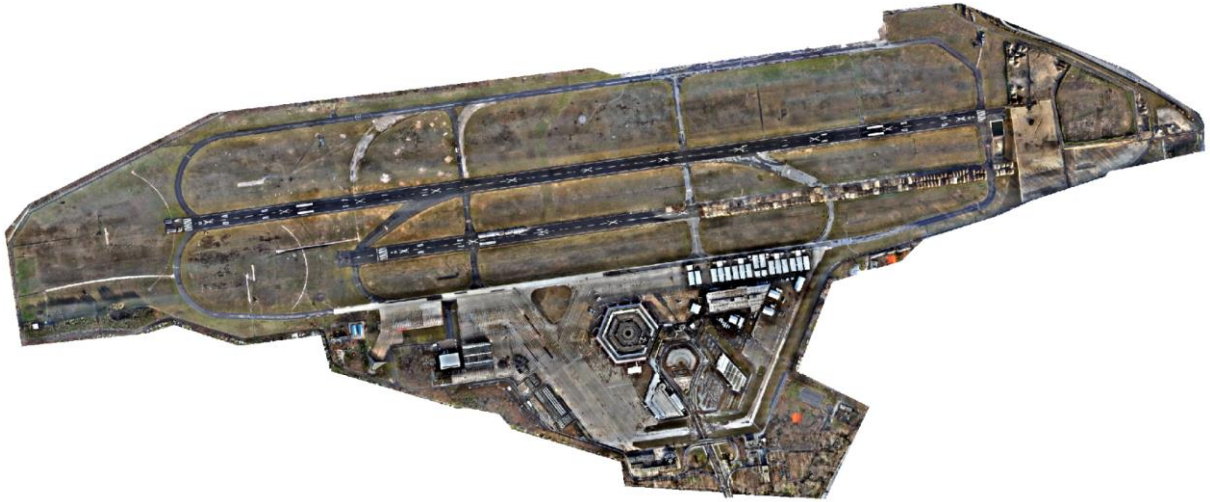


Abbildung 3-4: Erstelltes Orthomosaik vom 02.02.2023 (GSD 1,37cm)

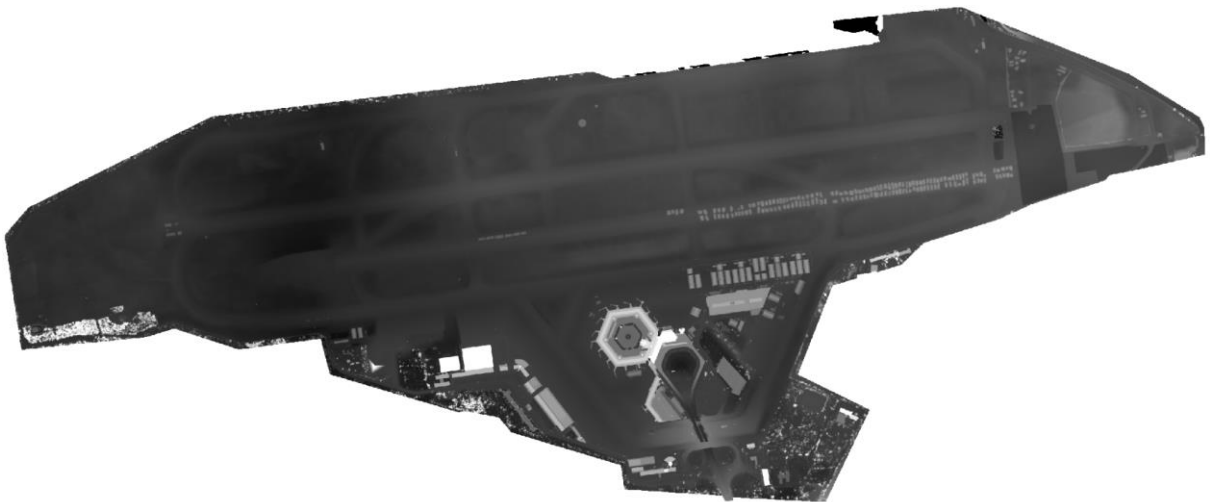


Abbildung 3-5: Erstelltes digitales Höhenmodell vom 02.02.2023

3.1.2 Erkennungsrate

Tabelle 2: Auflistung der Erkennungsraten von Flug1

GCP	erkennbar	erkannt	unerkannt	Erkennungsrate	Begründung
1	12	12	0	100,00%	
2	17	16	1	94,12%	Überbelichtung
3	23	22	1	95,65%	Überbelichtung & heller Untergrund
4	15	15	0	100,00%	
5	15	11	4	73,33%	Überbelichtung & heller Untergrund
6	23	23	0	100,00%	
7	15	15	0	100,00%	
8	12	12	0	100,00%	
9	15	15	0	100,00%	
10	15	15	0	100,00%	
11	10	10	0	100,00%	
12	15	15	0	100,00%	
13	14	14	0	100,00%	
14	12	12	0	100,00%	
15	16	16	0	100,00%	
16	14	14	0	100,00%	
17	15	15	0	100,00%	
18	15	12	3	80,00%	Schachtdeckel
19	19	19	0	100,00%	
20	15	15	0	100,00%	
21	19	19	0	100,00%	
22	20	20	0	100,00%	
23	14	14	0	100,00%	
Gesamt	360	351	9	97,50%	

Die erreichte Erkennungsrate beträgt insgesamt 97,50%, weil insgesamt 352 von 360 möglichen GCPs erkannt werden konnten. Die unerkannten GCPs sind zum einen auf ungünstige Lichtverhältnisse in Verbindung mit einem hellen Betonuntergrund zurückzuführen. Zum anderen wurde ein GCP auf einem kreisrunden Schacht verortet, was anscheinend die Erkennung beeinträchtigt.

3.1.3 Positionsgenauigkeit

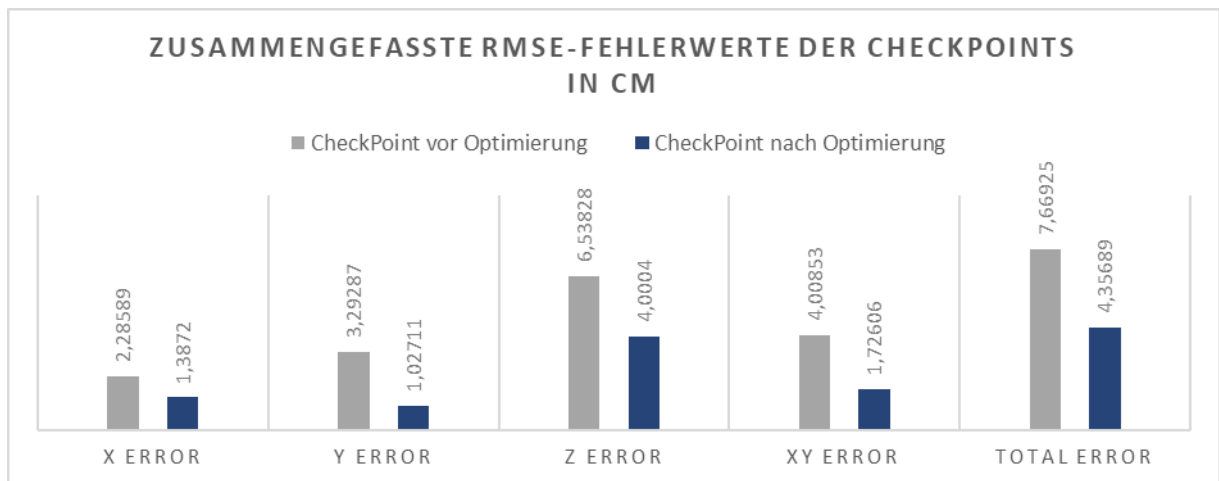


Abbildung 3-6: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur

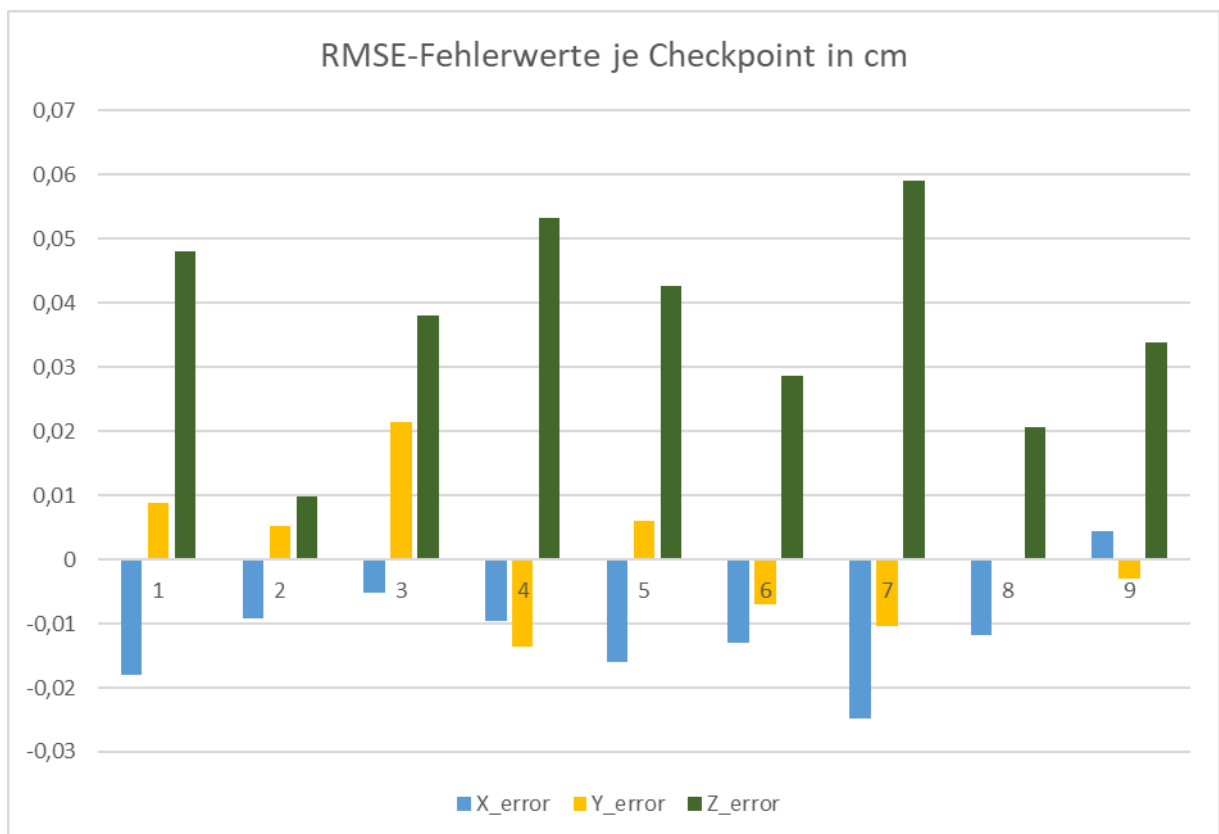


Abbildung 3-7: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint

Die Höhenfehler sind wie zu erwarten größer als die Lagefehler. Durch die Kameraoptimierung nach Korrektur der GCPs konnte die Genauigkeit signifikant verbessert werden. Die definierten Schwellenwerte für die Positionsgenauigkeit konnten eingehalten werden. ($XY < 3\text{cm}$; $Z < 5\text{cm}$)

3.2 Ergebnis Befliegung 2 (09.02.2023)

3.2.1 Orthomosaik und DEM



Abbildung 3-8: Erstelltes Orthomosaik vom 09.02.2023 (GSD 1,37cm)



Abbildung 3-9: Erstelltes digitales Höhenmodell vom 09.02.2023

3.2.2 Erkennungsrate

Tabelle 3: Auflistung der Erkennungsraten von Flug2

GCP	erkennbar	erkannt	unerkannt	Erkennungsrate	Begründung
1	15	15	0	100,00%	
2	16	16	0	100,00%	
3	24	7	17	29,17%	Schneedecke
4	15	15	0	100,00%	
5	15	15	0	100,00%	
6	23	5	18	21,74%	Schneedecke
7	15	15	0	100,00%	
8	13	13	0	100,00%	
9	14	14	0	100,00%	
10	14	14	0	100,00%	
11	10	0	10	0,00%	Schneedecke
12	15	1	14	6,67%	Schneedecke
13	15	15	0	100,00%	
14	18	17	1	94,44%	
15	15	0	15	0,00%	Schneedecke
16	15	0	15	0,00%	Schneedecke
17	15	15	0	100,00%	
18	15	7	8	46,67%	Schachtdeckel
19	21	21	0	100,00%	
20	10	10	0	100,00%	
21	20	20	0	100,00%	
22	21	21	0	100,00%	
23	17	17	0	100,00%	
Gesamt	371	273	98	73,58%	

Die erreichte Erkennungsrate beträgt insgesamt 73,58%, weil insgesamt 273 von 371 möglichen GCPs erkannt werden konnten.

Die Besonderheit bei dieser Bildbefliegung war eine partielle Schneedecke, welche einige der GCPs teilweise (Abbildung 3-10) oder sogar komplett verdeckt (Abbildung 3-11) hat. Teilweise verdeckte GCP's wurden mitunter nicht ganz korrekt erkannt, was sich vermutlich auch ungünstig auf die Positionsgenauigkeit ausgewirkt hat.

Auch in dieser Befliegung gab es Schwierigkeiten bei der Erkennung des auf dem Schachtdeckel verorteten GCPs.



Abbildung 3-10: teilweise vom Schnee verdeckter GCP



Abbildung 3-11: von einer dünnen Schneedecke verdeckter GCP

3.2.3 Positionsgenauigkeit

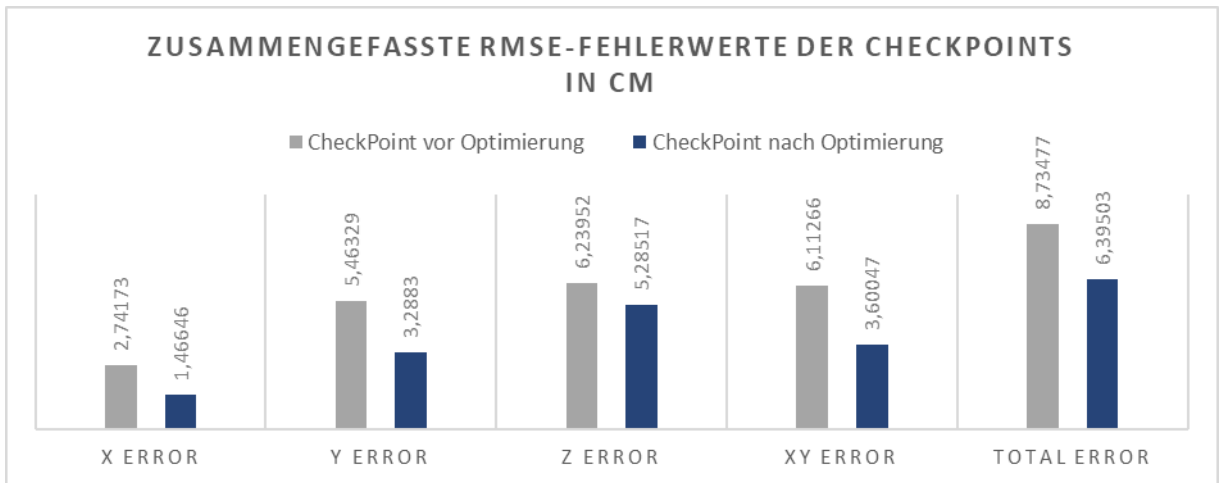


Abbildung 3-12: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur

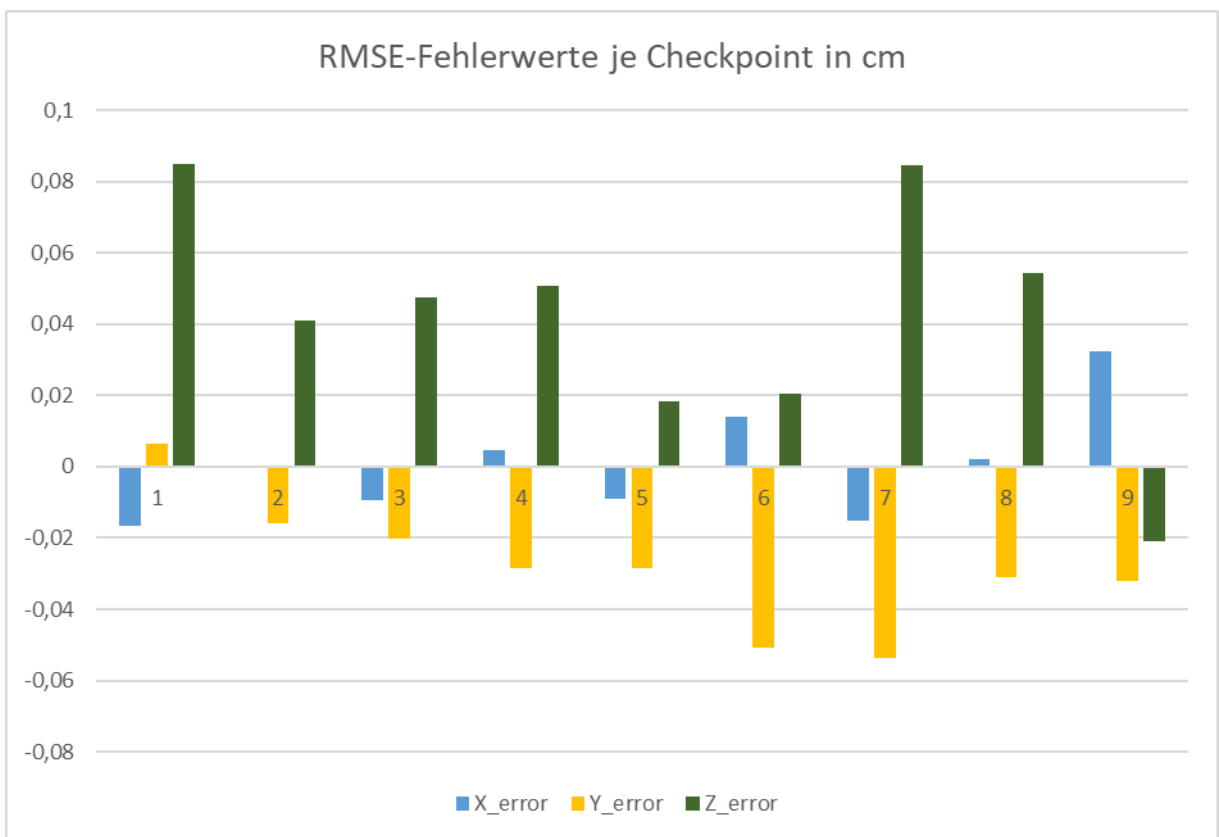


Abbildung 3-13: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint

Die Höhenfehler sind, wie zu erwarten, größer als die Lagefehler. Durch die Kameraoptimierung nach Korrektur der GCPs konnte die Genauigkeit signifikant verbessert werden. Die definierten Schwellenwerte für die Positionsgenauigkeit konnten hier nicht eingehalten werden. ($XY > 3\text{cm}$; $Z > 5\text{cm}$)

3.3 Ergebnis Befliegung 3 (16.02.2023)

3.3.1 Orthomosaik und DEM



Abbildung 3-14: Erstelltes Orthomosaik vom 16.02.2023 (GSD 1,37cm)

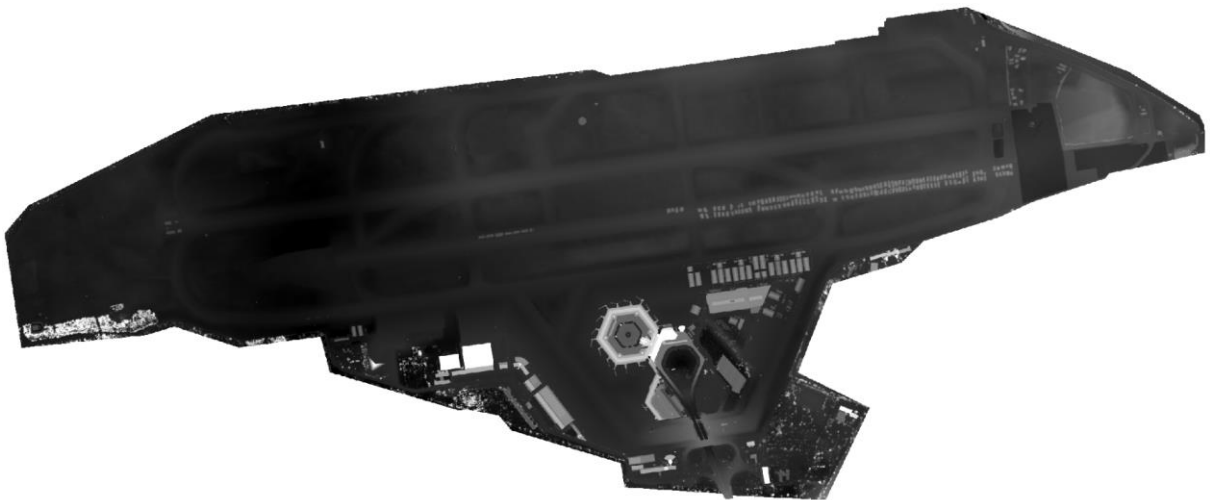


Abbildung 3-15: Erstelltes digitales Höhenmodell vom 16.02.2023

3.3.2 Erkennungsrate

Tabelle 4: Auflistung der Erkennungsraten von Flug3

GCP	erkennbar	erkannt	unerkannt	Erkennungsrate	Begründung
1	14	14	0	100,00%	
2	15	15	0	100,00%	
3	25	20	5	80,00%	Überbelichtung & heller Untergrund
4	14	14	0	100,00%	
5	15	15	0	100,00%	
6	25	10	15	40,00%	parkendes Auto
7	15	15	0	100,00%	
8	13	13	0	100,00%	
9	20	20	0	100,00%	
10	14	14	0	100,00%	
11	12	12	0	100,00%	
12	15	15	0	100,00%	
13	15	15	0	100,00%	
14	15	15	0	100,00%	
15	15	10	5	66,67%	Überbelichtung & heller Untergrund
16	14	14	0	100,00%	
17	15	15	0	100,00%	
18	15	11	4	73,33%	Schachtdeckel
19	20	20	0	100,00%	
20	15	15	0	100,00%	
21	17	17	0	100,00%	
22	20	20	0	100,00%	
23	16	16	0	100,00%	
Gesamt	374	345	29	92,25%	

Die erreichte Erkennungsrate beträgt insgesamt 92,25%, weil insgesamt 345 von 374 möglichen GCPs erkannt werden konnten. Die unerkannten GCPs sind zum einen auf ungünstige Lichtverhältnisse in Verbindung mit einem hellen Betonuntergrund zurückzuführen.

Auch in dieser Befliegung gab es Schwierigkeiten bei der Erkennung des auf dem Schachtdeckel verorteten GCPs. (siehe Abbildung 3-17)

Ein GCP wurde teilweise von einem parkenden PKW verdeckt, was dazu geführt hat, dass der GCP auf einigen Fotos nicht erkannt wurde, auf anderen trotz der Verdeckung jedoch erkannt wurde. (siehe Abbildung 3-16)

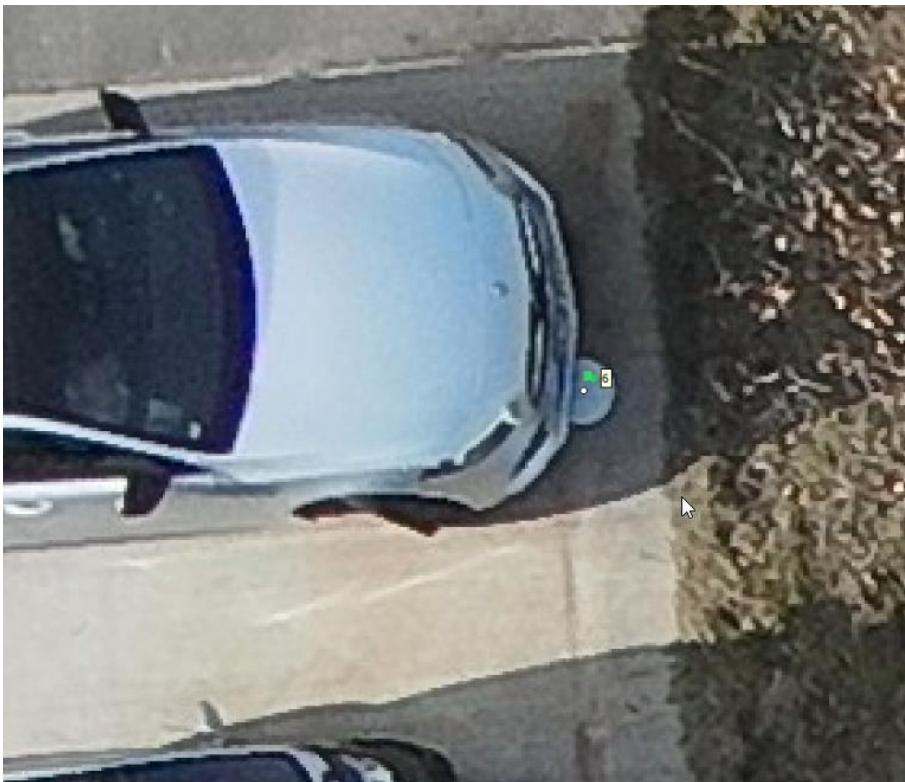


Abbildung 3-16: GCP erkannt, obwohl teilweise von Fahrzeug verdeckt

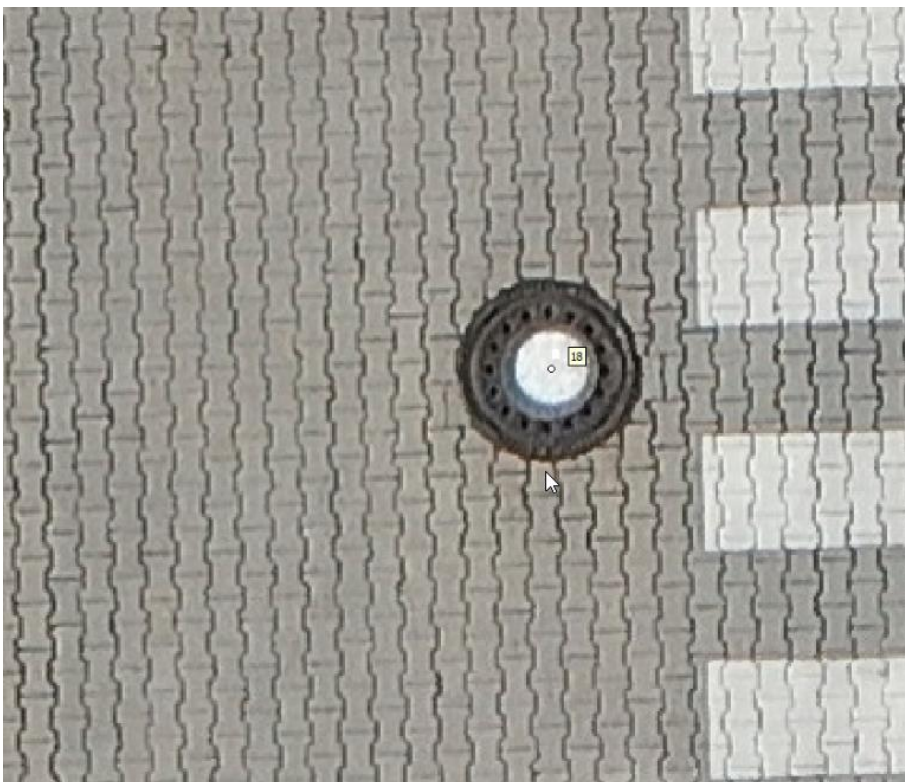


Abbildung 3-17: Der Schachtdeckel stellt für HCT eine Herausforderung dar

3.3.3 Positionsgenauigkeit

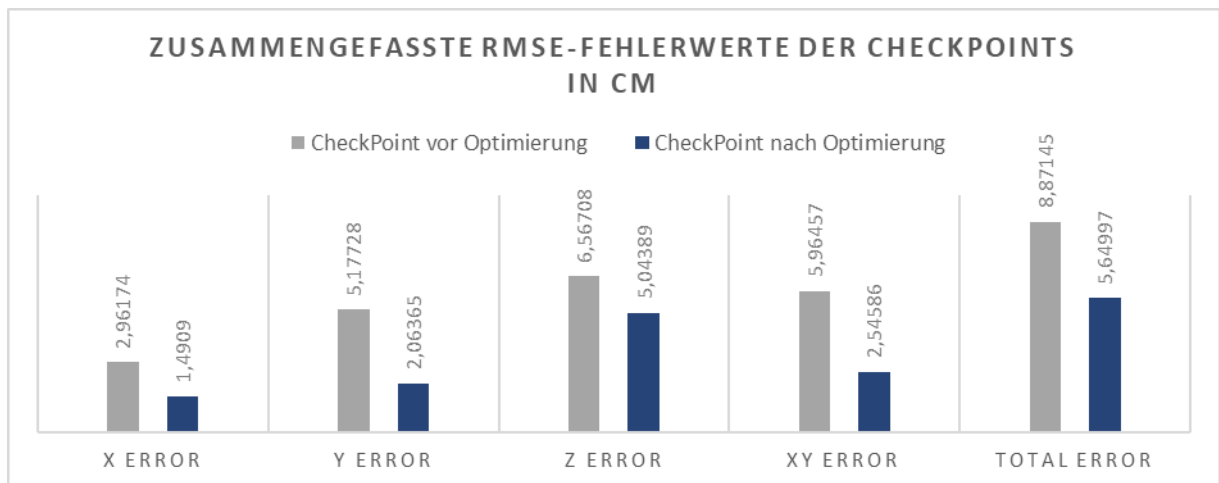


Abbildung 3-18: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur

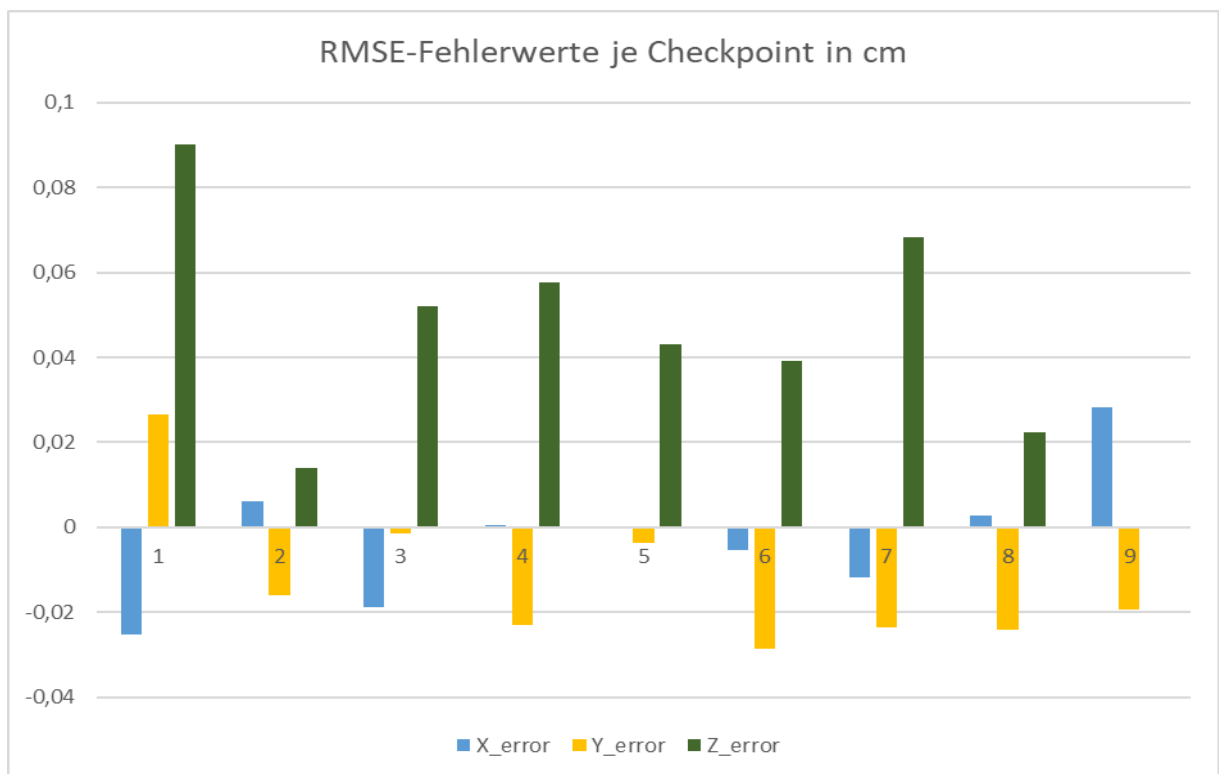


Abbildung 3-19: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint

Die Höhenfehler sind wie zu erwarten größer als die Lagefehler. Durch die Kameraoptimierung nach Korrektur der GCPs konnte die Genauigkeit signifikant verbessert werden. Die definierten Schwellenwerte für die Positionsgenauigkeit konnten in der Lage eingehalten werden und wurden in der Höhe leicht verfehlt. ($XY < 3\text{cm}$; $Z > 5\text{cm}$)

3.4 Ergebnis Befliegung 4 (23.02.2023)

3.4.1 Orthomosaik und DEM



Abbildung 3-20: Erstelltes Orthomosaik vom 23.02.2023 (GSD 1,37cm)



Abbildung 3-21: Erstelltes digitales Höhenmodell vom 23.02.2023

3.4.2 Erkennungsrate

Tabelle 5: Auflistung der Erkennungsraten von Flug4

GCP	erkennbar	erkannt	unerkannt	Erkennungsrate	Begründung
1	14	14	0	100,00%	
2	15	15	0	100,00%	
3	25	22	3	88,00%	Überbelichtung & heller Untergrund
4	15	15	0	100,00%	
5	14	14	0	100,00%	
6	23	23	0	100,00%	
7	15	15	0	100,00%	
8	14	14	0	100,00%	
9	15	15	0	100,00%	
10	14	14	0	100,00%	
11	10	10	0	100,00%	
12	16	16	0	100,00%	
13	15	15	0	100,00%	
14	15	15	0	100,00%	
15	15	15	0	100,00%	
16	15	15	0	100,00%	
17	16	16	0	100,00%	
18	15	11	4	73,33%	Schachtdeckel
19	21	21	0	100,00%	
20	15	15	0	100,00%	
21	17	17	0	100,00%	
22	21	21	0	100,00%	
23	16	16	0	100,00%	
Gesamt	371	364	7	98,11%	

Die erreichte Erkennungsrate beträgt insgesamt 98,11%, weil insgesamt 364 von 371 möglichen GCPs erkannt werden konnten. Die unerkannten GCPs sind in einem Fall auf ungünstige Lichtverhältnisse in Verbindung mit einem hellen Betonuntergrund zurückzuführen.

Auch in dieser Befliegung gab es teilweise Schwierigkeiten bei der Erkennung des auf dem Schachtdeckel verorteten GCPs.

3.4.3 Positionsgenauigkeit

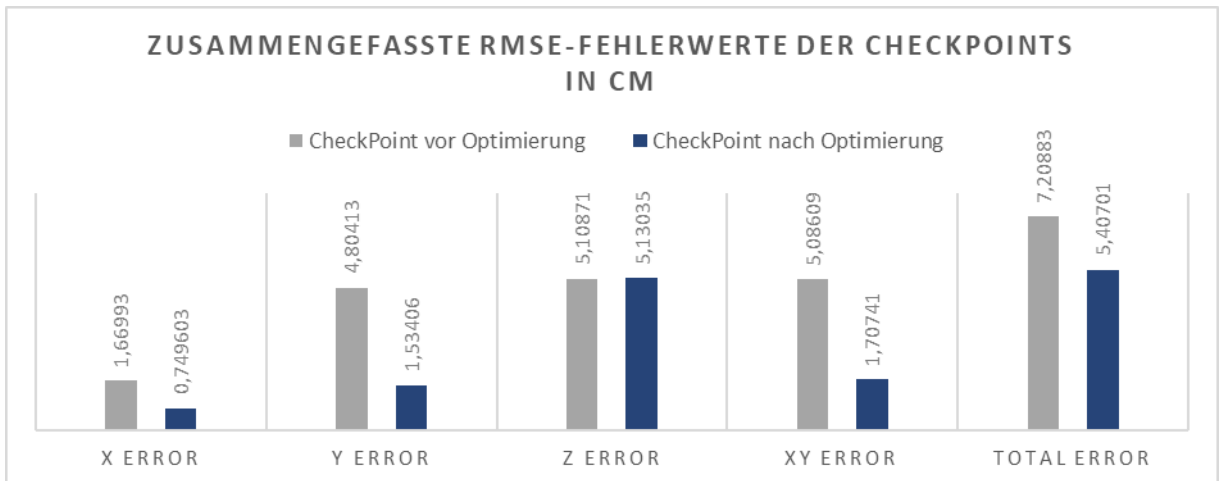


Abbildung 3-22: Vergleich der RMSE-Fehlerwerte vor und nach der GCP-Korrektur

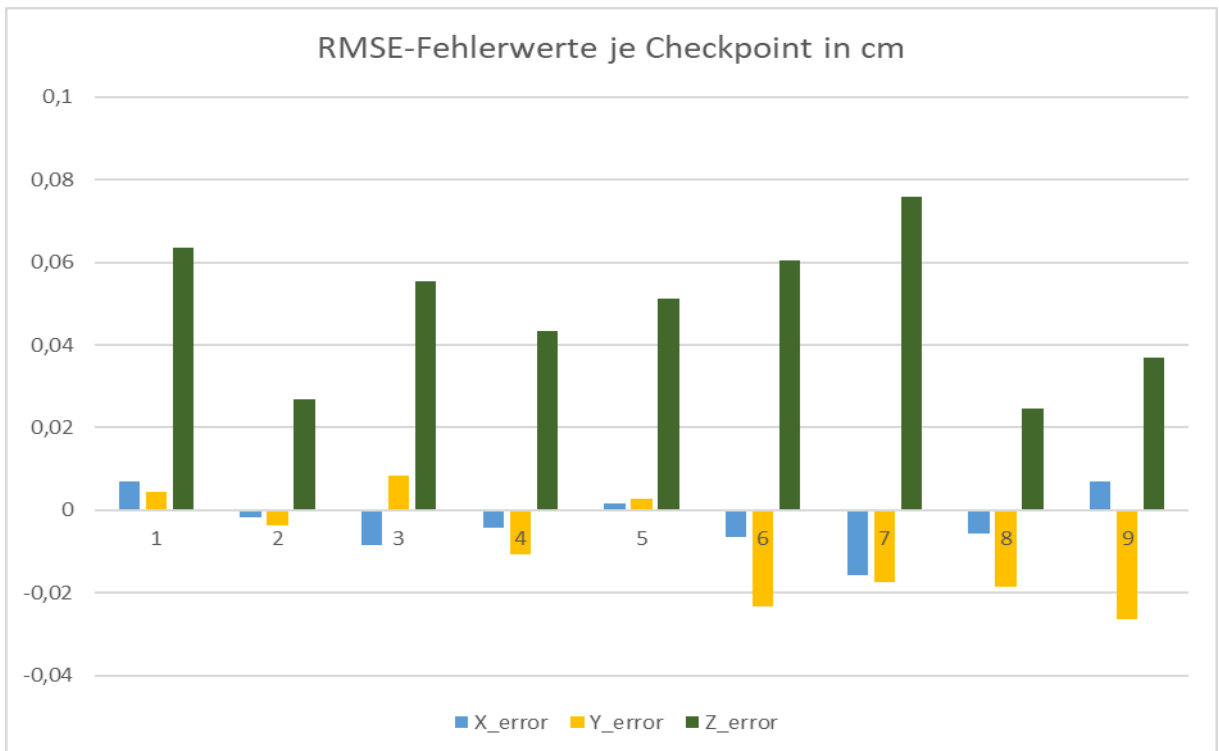


Abbildung 3-23: Vergleich der erreichten RMSE-Fehlerwerte pro Checkpoint

Die Höhenfehler sind wie zu erwarten größer als die Lagefehler. Durch die Kameraoptimierung nach Korrektur der GCPs konnte die Lagegenauigkeit signifikant verbessert werden, während die Höhengenaugigkeit sich leicht verschlechtert hat. Die definierten Schwellenwerte für die Positionsgenauigkeit konnten in der Lage eingehalten werden und wurden in der Höhe leicht verfehlt. ($XY < 3\text{cm}$; $Z > 5\text{cm}$)

3.5 Zusammenfassung

In diesem Abschnitt erfolgt eine Gegenüberstellung der Ergebnisse von allen vier Bildbefliegungen. Wie schon bei den Einzelergebnissen wird auch hier zwischen den Erkennungsraten und den Positionsgenauigkeiten unterschieden. Die Daten wurden tabellarisch bzw. grafisch aufbereitet.

3.5.1 Erkennungsraten

Tabelle 6: Erkennungsrate je Befliegung

Befliegung	erkennbar	erkannt	unerkannt	Erkennungsrate
02.02.2023	360	351	9	97,50%
09.02.2023	371	273	98	73,58%
16.02.2023	374	345	29	92,25%
23.02.2023	371	364	7	98,11%
Gesamt	1476	1333	143	90,36%

Die Gesamterkennungsrate aller Befliegungen beträgt 90,36%, weil von 1476 möglichen Erkennungen 1333 erkannt wurden. Die Erkennungsrate der Befliegung 2 ist mit 73,58% deutlich niedriger als die übrigen drei Befliegungen, welche durchweg eine Erkennungsrate größer 90% aufweisen. Der Grund für die schlechte Erkennungsrate in Befliegung 2 ist die Beeinträchtigung durch Schneeablagerungen.

Die folgende Abbildung zeigt die Erkennungsraten aller GCPs je Befliegung:

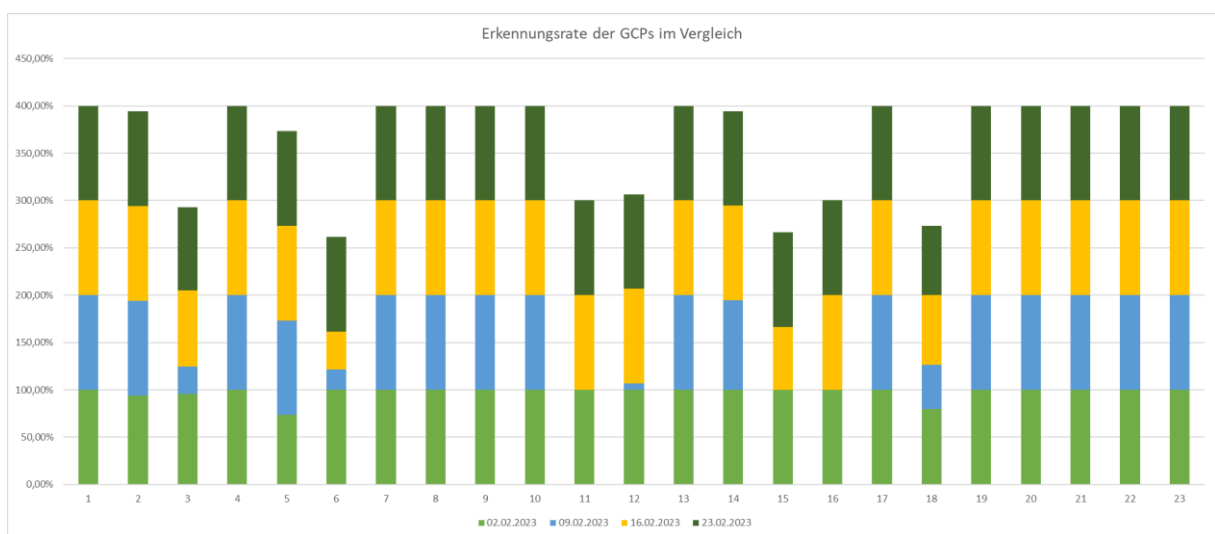


Abbildung 3-24: Visualisierung der Erkennungsraten aller GCPs je Befliegung

3.5.2 Positionsgenauigkeiten

Die folgende Abbildung zeigt den RMSE der Checkpoints unterteilt nach Lage (XY) und Höhe (Z) für alle vier Bildbefliegungen:

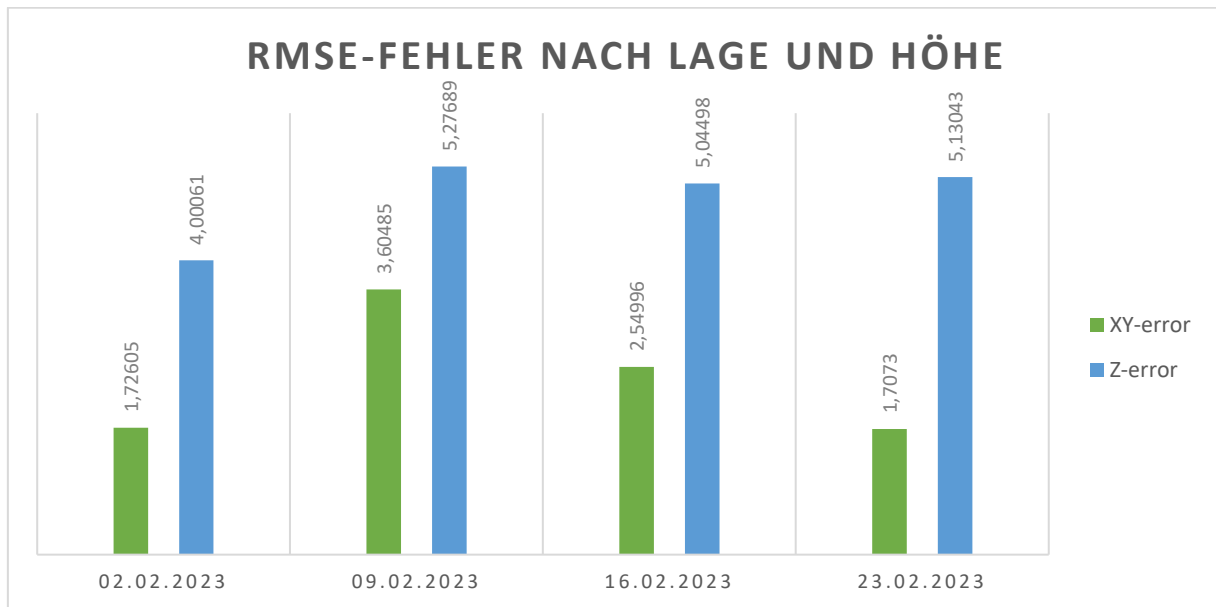


Abbildung 3-25: Visualisierung der Gesamtfehlerwerte je Befliegung unterteilt nach Lage und Höhe

Die nächste Abbildung zeigt die Aufsummierten RMSE-Werte alle Befliegungen für jeden Checkpoint. Die Höhenfehler sind in Summe größer als die Lagefehler. Die Höhenfehler von CP1 und CP7 sind auffällig größer als die Höhenfehler der anderen Checkpoints.

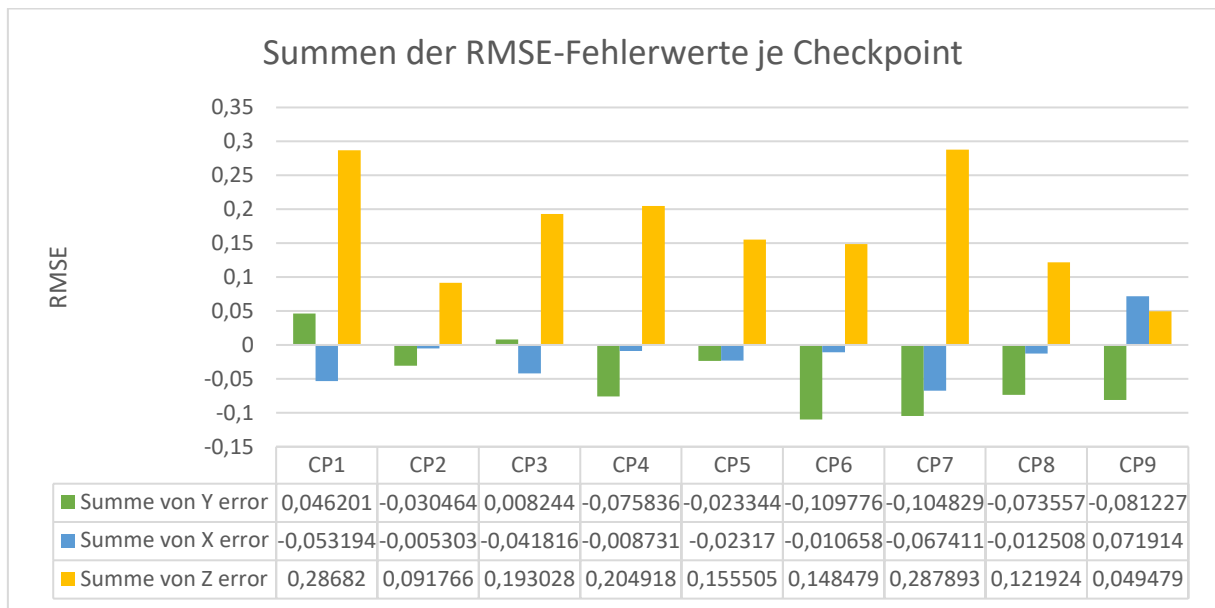


Abbildung 3-26: Visualisierung der summierten Fehlerwerte je Checkpoint unterteilt nach Lage und Höhe

Dieses Diagramm beschreibt die RMSE-Mittelwerte jeweils vor und nach der Optimierung durch die GCPs und belegt damit die Verbesserung der Positionsgenauigkeit:

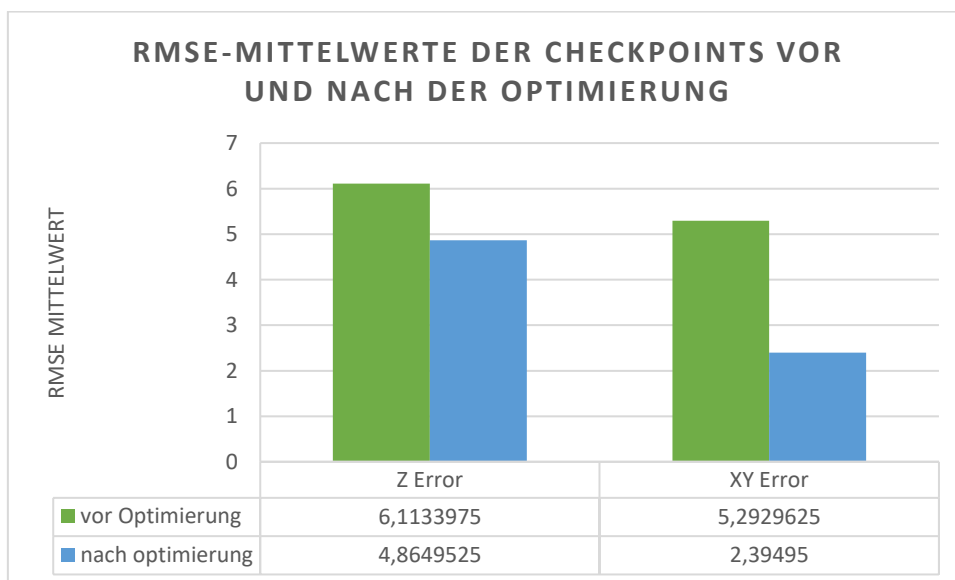


Abbildung 3-27: Darstellung der mittleren Fehlerwerte der Checkpoints vor und nach der Optimierung durch GCPs

4 Diskussion

In diesem Kapitel erfolgt die Auswertung der in Kapitel 3 vorgestellten Ergebnisse.

Dafür sollte zunächst die besondere Rolle von Befliegung 2 hervorgehoben werden. Da in dieser Befliegung einige GCPs vom Schnee ganz oder teilweise bedeckt waren, wurden sowohl bei den Erkennungsraten als auch bei den Positionsgenauigkeiten die schlechtesten Ergebnisse im Vergleich erzielt. Die angestrebten Positionsgenauigkeiten von mindestens 3cm in Lage und 5cm in Höhe wurden in dieser Befliegung jeweils knapp verfehlt. Diese Erkenntnis bestätigt jedoch, dass sich die implementierte Methode positiv auf die Verbesserung der Positionsgenauigkeit auswirkt. In jedem Fall stellt Schnee durch seine überdeckenden Eigenschaften eine besondere Komponente bei der Verortung von stationären GCPs dar. Aufgrund der relativ wenigen Schneetage in Berlin wurde dieses Szenario im Projekt bisher bewusst nicht berücksichtigt und besitzt daher nur eine eingeschränkte Repräsentationskraft bezüglich Erkennungsrate und Positionsgenauigkeit.

Die Automatisierung des Gesamtprozesses lief für alle vier Prozesse reibungslos ab. Der Prozess wurde jeweils am Donnerstagnachmittag gestartet und sonntags im Laufe des Tages fertiggestellt, sodass alle Projektbeteiligten am Montag in der Früh über das Geoportal Zugriff auf das Orthomosaik hatten. Die Zeitersparnis gegenüber der manuellen Auswertung liegt, bedingt durch die nahtlose Aneinanderreihung der Teilprozesse, bei 2 bis 3 Tagen. Die Einsparung von Arbeitszeit beträgt wöchentlich ca. 1,5h und ist vor allem auf die nun automatische Erkennung der GCPs zurückzuführen. Was den Zeitbedarf anbelangt, so nahm letztere jeweils weniger als 5 Minuten Berechnungszeit in Anspruch und fällt damit bei der Gesamtdauer der Berechnungszeit kaum ins Gewicht.

4.1 Erkennungsrate

Betrachtet man alle vier Befliegungen zusammen, so wurde eine mittlere GCP-Erkennungsrate von **90,36%** erreicht. Wie oben beschrieben wird dieses Ergebnis durch die Schneedecke in Befliegung 2 negativ beeinflusst. Schnee ist somit insgesamt der Hauptgrund für nicht erkannte GCPs. Würde Befliegung 2 nicht in den Ergebnissen berücksichtigt, so würde die mittlere Erkennungsrate auf **95,95%** steigen. Die Ursachen für die restlichen **4,05%** unerkannter GCPs konnten grundsätzlich ausgemacht und in drei Kategorien unterteilt werden:

1. Kategorie: GCP auf Schachtdeckel (GCP Nr. 18)

Dieser GCP wurde auf einem runden Schachtdeckel (siehe Abbildung 4-1) realisiert und ist nach Schnee die zweithäufigste Ursache für nicht erkannte GCPs. Bei jeder Befliegung gab es Probleme mit der Erkennung dieses GCPs. Die Erkennungsraten lagen zwischen 46% – 80%. Ein Schachtdeckel ist grundsätzlich nicht für die Platzierung eines GCP geeignet. Da der GCP nicht mittig aufgebracht wurde, würde diesem Fall eine Verdrehung des Deckels reichen, um das Ergebnis negativ zu beeinflussen. Im Zuge dieser Arbeit wurde der GCP bewusst auf dem Deckel aufgebracht, um die Robustheit und die Grenzen der implementierten Methode zu testen. Die erzielten Ergebnisse bestätigen, dass die HCT-Methode, vermutlich durch die zahlreichen geschachtelten Kreisstrukturen, in solch speziellen Szenarien nicht zuverlässig arbeitet.

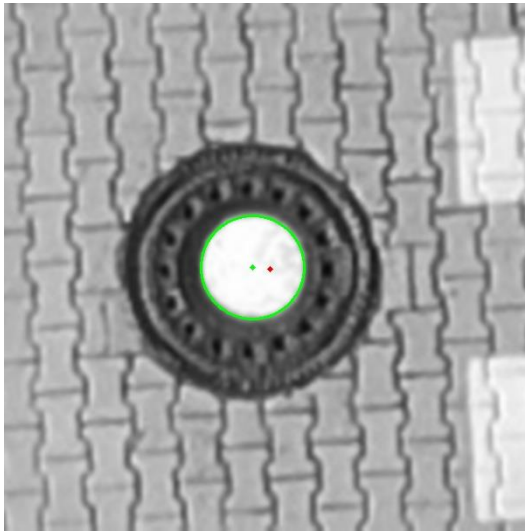


Abbildung 4-1: erkannter GCP auf Schachtdeckel

2. Kategorie: Überbelichtung & heller Untergrund

Die GCPs 3 und 5 wurden im Gegensatz zu den anderen GCPs nicht auf Asphalt, sondern auf etwas hellerem Betonuntergrund aufgebracht. Dadurch ist der Kontrast zwischen GCP und Untergrund bei GCP 3 und 5 geringer. Kommen noch ungünstige Lichtverhältnisse, wie eine morgendliche Wintersonne sowie Raureif dazu, verringert sich der Kontrast nochmals, was zu einigen unerkannten GCPs geführt hat. Die folgende Abbildung zeigt das Beispiel eines vermutlich nur knapp erkannten GCP 3 aus Befliegung 2:

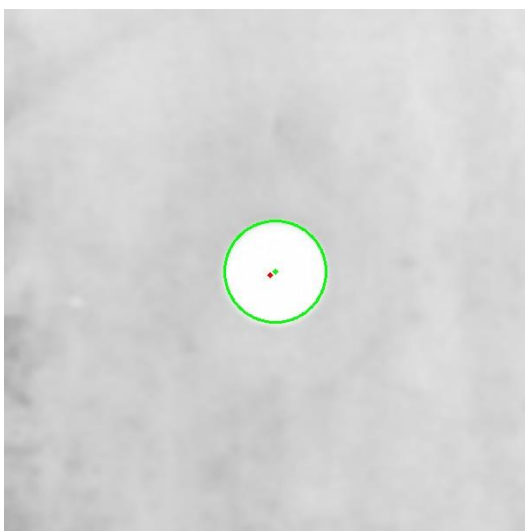


Abbildung 4-2: Erkannter GCP 3 bei schlechten Lichtverhältnissen

3. Kategorie: geparktes Fahrzeug

Während Befliegung 3 wurde GCP 6 teilweise durch ein geparktes Fahrzeug verdeckt. Dadurch konnte der GCP auf 15 Fotos nicht erkannt werden.

Ein GCP wurde vermutlich auf einem Foto durch eine leichte Unsauberkeit beim Farbauftrag in Befliegung 1 nicht erkannt und keiner Kategorie zugeordnet.

Insgesamt kann festgehalten werden, dass die Erkennung der Kreise sehr zuverlässig funktioniert. Die Ursachen für alle nicht erkannten Fälle konnten sicher hergeleitet werden. Gegen Kategorie 1 und 2 können gezielt Maßnahmen ergriffen bzw. Fehler gleicher Art zukünftig vermieden werden. Gegen Kategorie 3 kann nur bedingt etwas unternommen werden, da der Ort nicht als Parkfläche vorgesehen ist.

Unter guten Bedingungen und mit Vermeidung von Fehlern der Kategorie 1 und 2 ist eine Erkennungsrate von 98% oder sogar 99% also durchaus realistisch. Weitere Fehlerquellen wurden im Rahmen dieser Arbeit nicht ermittelt, werden in den kommenden Befliegungen jedoch sehr wahrscheinlich auftreten.

4.2 Positionsgenauigkeit

Wie bereits beschrieben nimmt Befliegung 2 auch bei der Positionsgenauigkeit eine besondere Rolle ein. So wurden die eingangs festgelegten Schwellenwerte für Lage- und Höhengenaugigkeit um $\sim 6\text{mm}$ bzw. $\sim 3\text{mm}$ in dieser Befliegung überschritten.

In den drei anderen Befliegungen konnte die geforderte Lagegenauigkeit von 3cm eingehalten werden. Anders verhält es sich mit den Höhengenaugigkeiten. Hier konnte der geforderte Wert von 5cm lediglich in Befliegung 1 eingehalten werden. In den anderen drei Befliegungen wird dieser Wert,

wenn auch sehr knapp, um 0.5mm bis 3mm überschritten. Die Ursachen dafür können vielschichtig sein. Einerseits ist ein wesentlicher Grund wahrscheinlich auf die Parametrierung der Befliegungen zurückzuführen. Wie unter anderen Kersten, Schlömer et al. (2020) und Przybilla, Reuber et al. (2015) ausführen, ist eine Kreuzbefliegung zur Verbesserung der Aerotriangulation obligatorisch. Wie vorhergehend in Kapitel 1.2 dargelegt, wurde aus ökonomischen Gründen auf Kreuzbefliegungen verzichtet. Andererseits könnte die implementierte HCT-Methode ggf. auch für die Erkennung von GCPs auf Oblique-Aufnahmen erweitert werden. Die im Rahmen dieser Arbeit entwickelte Methode betrachtet lediglich Nadir-Aufnahmen, welche überwiegend vorliegen. Die Erkennung von Kreisen auf Oblique-Aufnahmen würde viele zusätzliche Herausforderungen schaffen, da die Kreise perspektivisch bedingt verzerrt werden und als Ellipsen erkennbar sind (siehe Abbildung 4-3 und Abbildung 4-4). Dieser Effekt variiert je nach Position des Kreises auf dem Bild und ist zudem auch abhängig vom Aufnahmewinkel und der Aufnahmehöhe. Es wäre also viele Parameter und Anpassungen für eine Berücksichtigung von Schrägaufnahmen notwendig. Vor einer Implementierung sollte der mögliche Benefit zunächst durch manuelles Referenzieren eruiert werden. Auch das Design der GCP's könnte für diesen Zweck überarbeitet werden. So könnte man statt der Farbkreise beispielsweise ballförmige Targets verwenden, wie sie auch beim Laserscanning zum Einsatz kommen. Dadurch würde sich zumindest die perspektivisch bedingte Verzerrung umgehen lassen. Gleichzeitig würde dies jedoch auch einen erhöhten Aufwand für die Erstellung und Pflege der GCPs bedeuten.



Abbildung 4-3: Vergleich von Ausschnitten aus Oblique-Aufnahmen mit einem Aufnahmewinkel von 45° bei ca. 90m Flughöhe. Der linke Ausschnitt befindet sich am oberen Bildrand und weist dadurch eine Höhere Verzerrung als der rechte Bildausschnitt auf, welcher sich eher am unteren Bildrand befindet. Zudem besteht die Darstellung aufgrund der Entfernung aus weniger Pixeln.

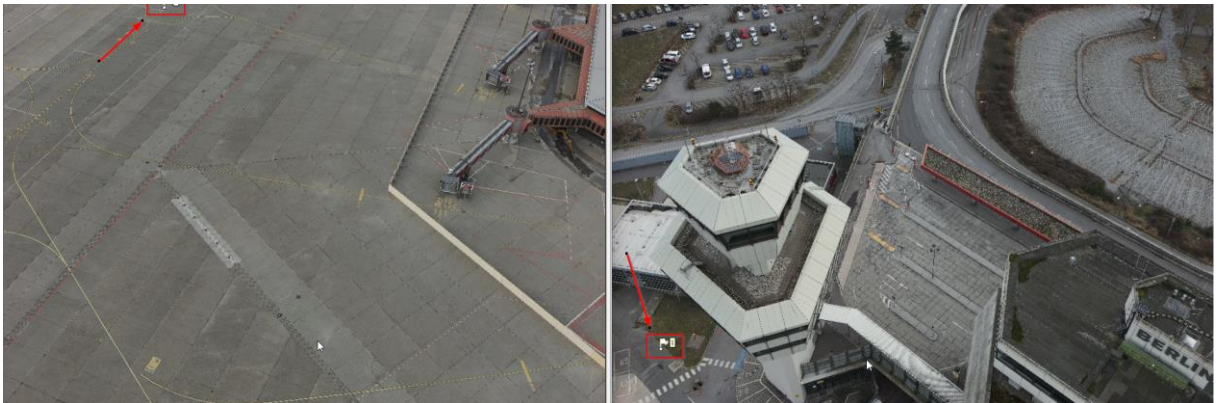


Abbildung 4-4: Verkleinerte Originalaufnahmen zu Abbildung 4-3

Ein weiterer Grund für das knappe Verfehlen der geforderten Höhengengenauigkeit könnte auch an der verwendeten Messmethodik zur Bestimmung der GCP- und Checkpoint-Positionen zu finden sein. Alle Referenzpunkte wurden mit ein und demselben Gerät bestimmt. Eine Überprüfung des Gerätes, beispielsweise über amtliche Referenzpunkte, könnte Aufschluss über eine mögliche Messungengenauigkeit geben. Weiterhin gäbe es noch präzisere Methoden zur Bestimmung der Positionen. So bietet der Markt höherpreisige GNSS-Messgeräte, welche eine noch bessere Positionsgenauigkeit versprechen. Bei dem verwendeten Leica FLX100 handelt es sich lediglich um ein Gerät der Einstiegsklasse. Zudem wäre die Bestimmung durch hochpräzise,

althergebrachte Methoden eine Variante. Die Lageposition könnte tachymetrisch und die Höhenposition durch ein Nivellement bestimmt werden. Mit dem notwendigen Sachverstand und Sorgfalt durchgeführt, wäre diese Art der Positionsbestimmung sicherlich präziser, wenngleich auch sehr viel aufwändiger.

Ein letztes Augenmerk bezüglich der Höhengenaugigkeit betrifft die Fehlerwerte der einzelnen Checkpoints. Auffällig sind die beiden Punkte CP1 sowie CP7. Für beide Checkpoints sind die ermittelten Höhenfehler durchgängig größer als bei den übrigen Checkpoints. Die Messreihen reichen nicht aus, um einen systematischen Fehler auszumachen, die Tendenzen deuten allerdings in diese Richtung. Nichtsdestotrotz sollten diese beiden Punkte vor weiteren Messungen überprüft werden.

Die optischen Vergleiche mittels ArcGIS Pro zeigten, dass nur bei kleineren Maßstäben wie 1:20 ein teilweise ein minimaler Versatz erkennbar sein kann (siehe Abbildung 4-5). Bei größeren Maßstäben wie 1:100 oder größer ist kein Versatz erkennbar, was auch beispielhaft in Abbildung 4-6 dargestellt ist. Der Abgleich mit einem grafischen Bestandsplan aus der Vermessung führte zu einem ähnlichen Ergebnis und wurde beispielhaft in Abbildung 4-7 dargestellt.



Abbildung 4-5: Beispiel Vergleich Befliegung 1 (links) mit Befliegung 4 (rechts) bei einem Maßstab von 1:20 über SWIPE-Tool in ArcGIS Pro. Ein minimaler Versatz ist erkennbar.

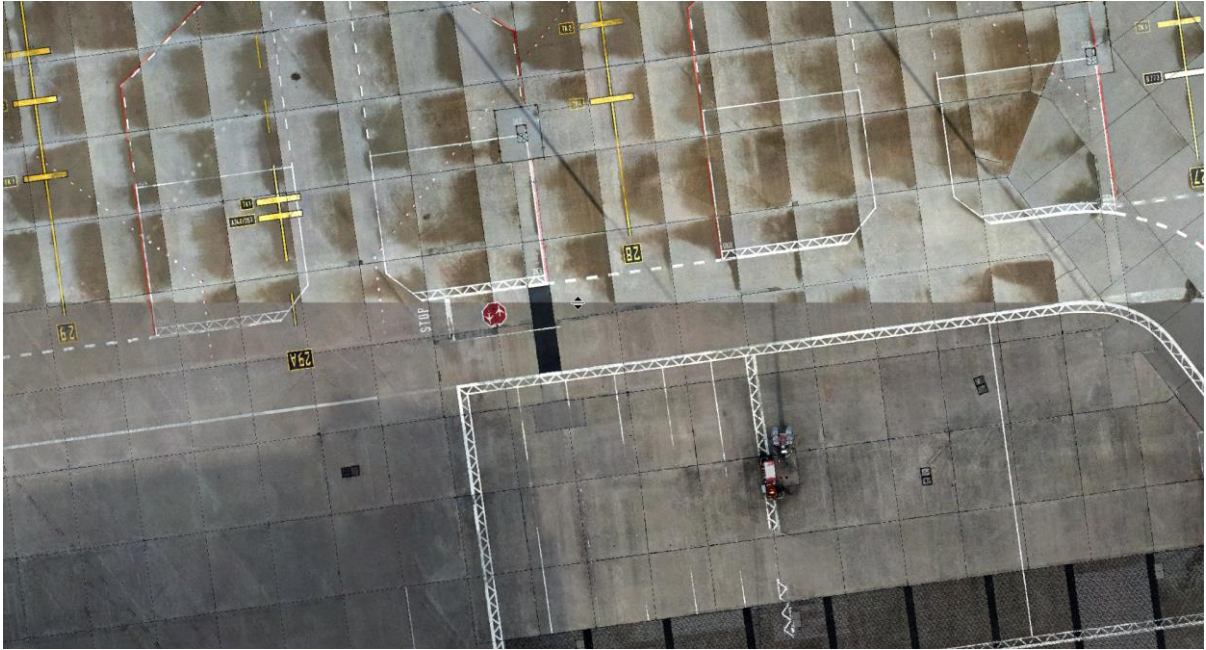


Abbildung 4-6: Beispiel Vergleich Befliegung 1 (oben) mit Befliegung 4 (unten) bei einem Maßstab von 1:200 über das SWIPE-Tool in ArcGIS Pro. Bei diesem Maßstab ist kein Versatz erkennbar.

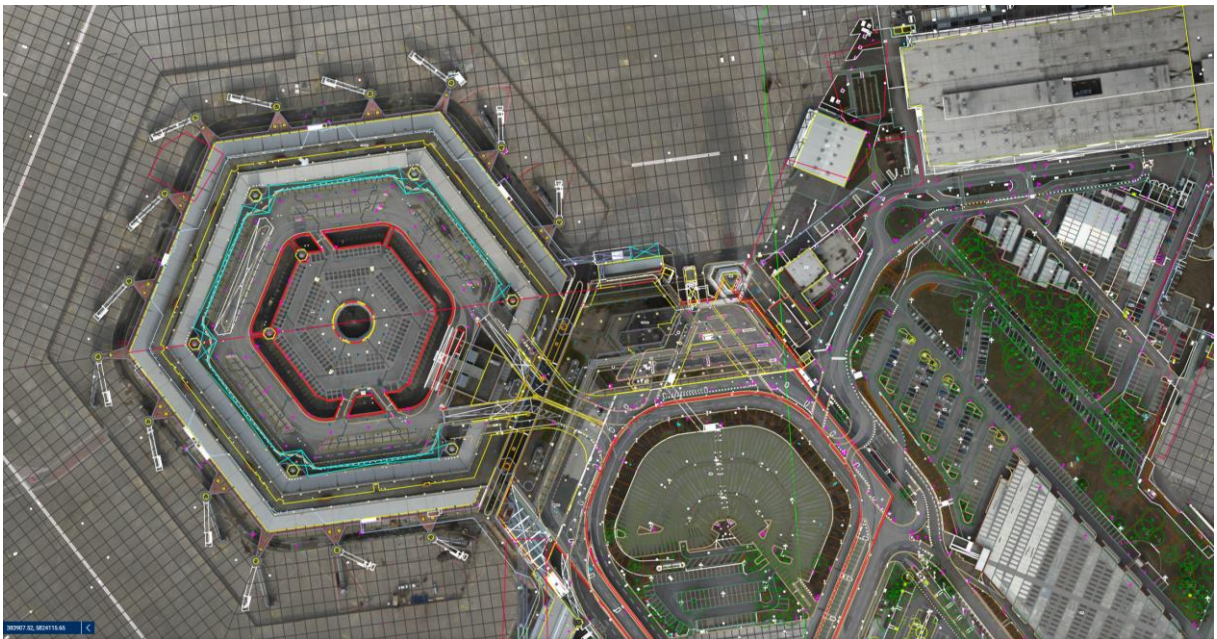


Abbildung 4-7: Darstellung von graphischem Bestandsplan über Orthomosaik von Befliegung 4

Zu erwähnen bleibt, dass es bei allen vier Befliegungen zeitweise Ausfälle der RTK-Verbindung gegeben hat. Dies kommt häufiger vor und im Normalfall unterbrechen die Geräte dann sofort die Befliegung so lange bis wieder eine stabile RTK-Verbindung besteht. Leider funktioniert dies nicht immer fehlerfrei, sodass es häufiger Bereiche mit Aufnahmen ohne RTK-Korrektur

gibt. Diese Bereiche können in den Metashape Protokollen im Anhang nachvollzogen werden. Ein Auszug aus dem Protokoll der Befliegung vier ist in der folgenden Abbildung dargestellt:

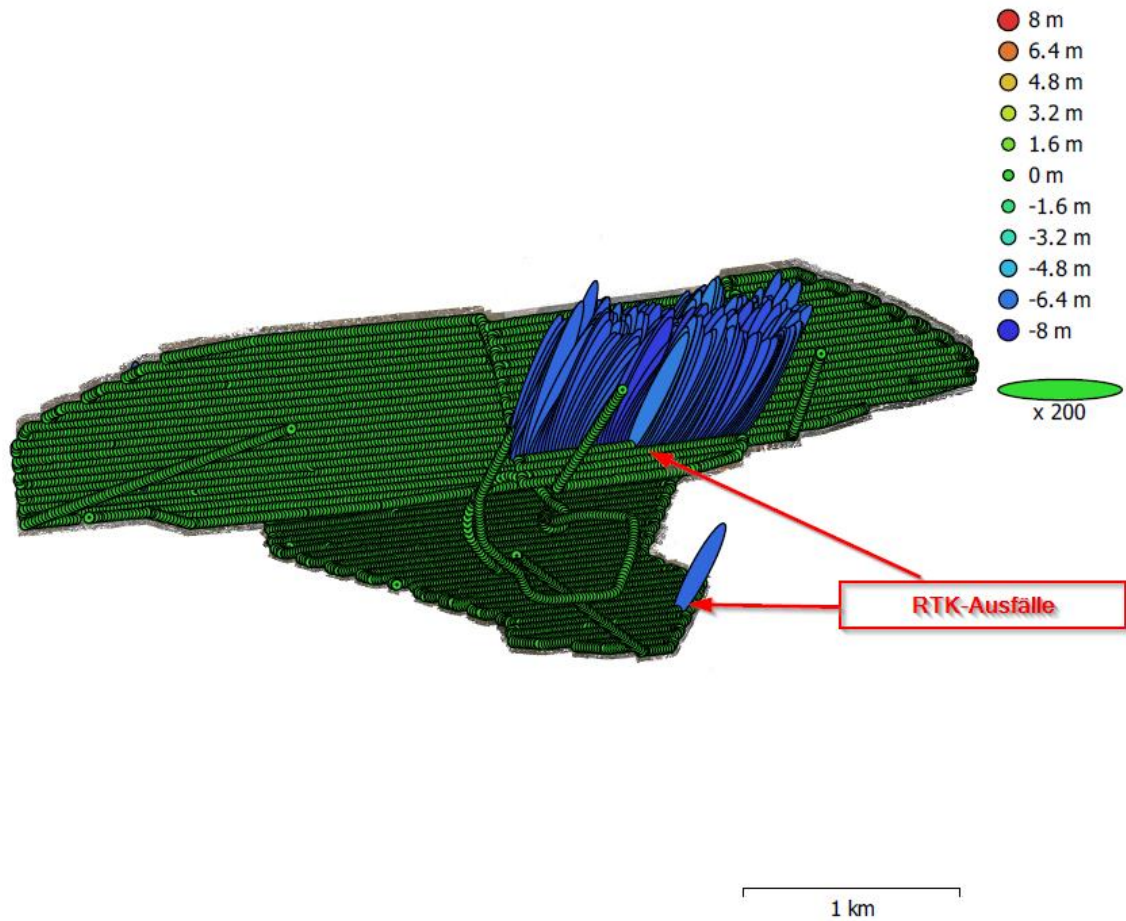


Fig. 4. Camera locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated camera locations are marked with a black dot.

Abbildung 4-8: Auszug aus dem Metashape Protokoll mit Angaben zu den Positionsgenauigkeiten der Fotos

5 Schlussfolgerung und Ausblick

Die Durchführung und die Ergebnisse dieser Arbeit haben gezeigt, dass die automatisierte Datenauswertung von UAV-Bildflügen von einzelnen Fotos bis hin zu einem Orthomosaik durch den Einsatz von RTK und automatischer Erkennung von Ground Control Points mittels Hough Circle Transform möglich ist. Die selbstdefinierte Positionsgenauigkeit von mindestens 3cm in der Lage konnte trotz einfacher Streifenbefliegung klar eingehalten werden. Anhand der an vier unterschiedlichen Tagen durchgeführten Befliegungen wurde aufgezeigt, dass der mittlere Lagefehler von $\sim 5,3\text{cm}$ ohne GCPs mithilfe der durch HCT erkannten GCPs auf $\sim 2,4\text{cm}$ reduziert werden konnte.

Die selbstdefinierte Höhengenaugigkeit von mindestens 5cm wurde in drei von vier durchgeführten Befliegungen nur um wenige Millimeter verfehlt und stellt damit aus Sicht des Autors trotzdem ein gutes Ergebnis dar. Der mittlere Höhenfehler aller vier Befliegungen konnte mithilfe der HCT-Methode von $\sim 6,1\text{cm}$ auf $\sim 4,9\text{cm}$ reduziert werden und würde bei gesamtheitlicher Betrachtung sogar unterhalb der definierten Fehlertoleranz liegen.

Die Möglichkeiten zur weiteren Verbesserung der Positionsgenauigkeit wurden aufgezeigt und sind wahrscheinlich in erster Linie bei einer Anpassung der Befliegungsparameter, wie beispielsweise der Durchführung einer Kreuzbefliegung zu finden. Negativen Einfluss hatte aller Wahrscheinlichkeit auch der teilweise Ausfall von RTK. Wie groß dieser Einfluss letztendlich war, kann nicht genau quantifiziert werden und müsste in einer separaten Untersuchung beleuchtet werden. Alternativ könnte zukünftig statt des RTKs auf PPK (Post-Processing Kinematic) zurückgegriffen werden, was in der Theorie die Befliegung vereinfachen und die Positionsgenauigkeit nach Angaben des LGB (2021) um 1cm bis 2cm verbessern könnte. Inwiefern sich diese Annahmen bestätigen und wie sich PPK gegebenenfalls in den automatischen Workflow integrieren lässt, wird der Autor dieser Arbeit zukünftig definitiv näher betrachten.

Die Erkennung von GCPs, in diesem Fall weiße Farbkreise mit einem Durchmesser von 30cm, aufgenommen aus Flughöhen von 90m bzw. 120m, mittels HCT hat sich als robust herausgestellt. Es wurde eine mittlere Erkennungsrate von 90,36% erzielt. Hervorzuheben ist dabei die Besonderheit von Befliegung 2, da an diesem Tag mehrere GCPs von Schnee verdeckt waren und somit nicht erkannt werden konnte. Würde man Befliegung 2 bei der Ergebnisbetrachtung außeracht lassen, so würden die übrigen drei Befliegungen eine Erkennungsrate von 95,95% erreichen. Die Gründe für die letzten ~4% nicht erkannter GCPs konnten vollständig eruiert werden und sind vor allem auf eine suboptimale Verortung teilweise kombiniert mit ungünstigen Lichtverhältnissen zurückzuführen. Durch die gewonnenen Erkenntnisse kann in Zukunft beides verhindert werden. Eine Verbesserung der Methode könnte ggf. durch die zusätzliche Erkennung von GCPs auf Schrägaufnahmen (Oblique) herbeigeführt werden. Die Umsetzung dieser Idee wäre durch perspektivbedingte Verzerrungen der Kreise zu Ellipsen und dynamische Entfernungen jedoch alles andere als Trivial.

Was den Zeitfaktor anbelangt, so konnte die Automatisierung den Gesamtprozess um zwei bis drei volle Tage verkürzen. Dies wurde durch die nahtlose Verkettung der einzelnen Arbeitsschritte ermöglicht. Nach allen vier Befliegungen war das Orthomosaik bereits am Montagmorgen für alle Projektbeteiligten im Geoportal verfügbar. Bei manueller Bearbeitung war dies zumeist erst am Mittwoch möglich.

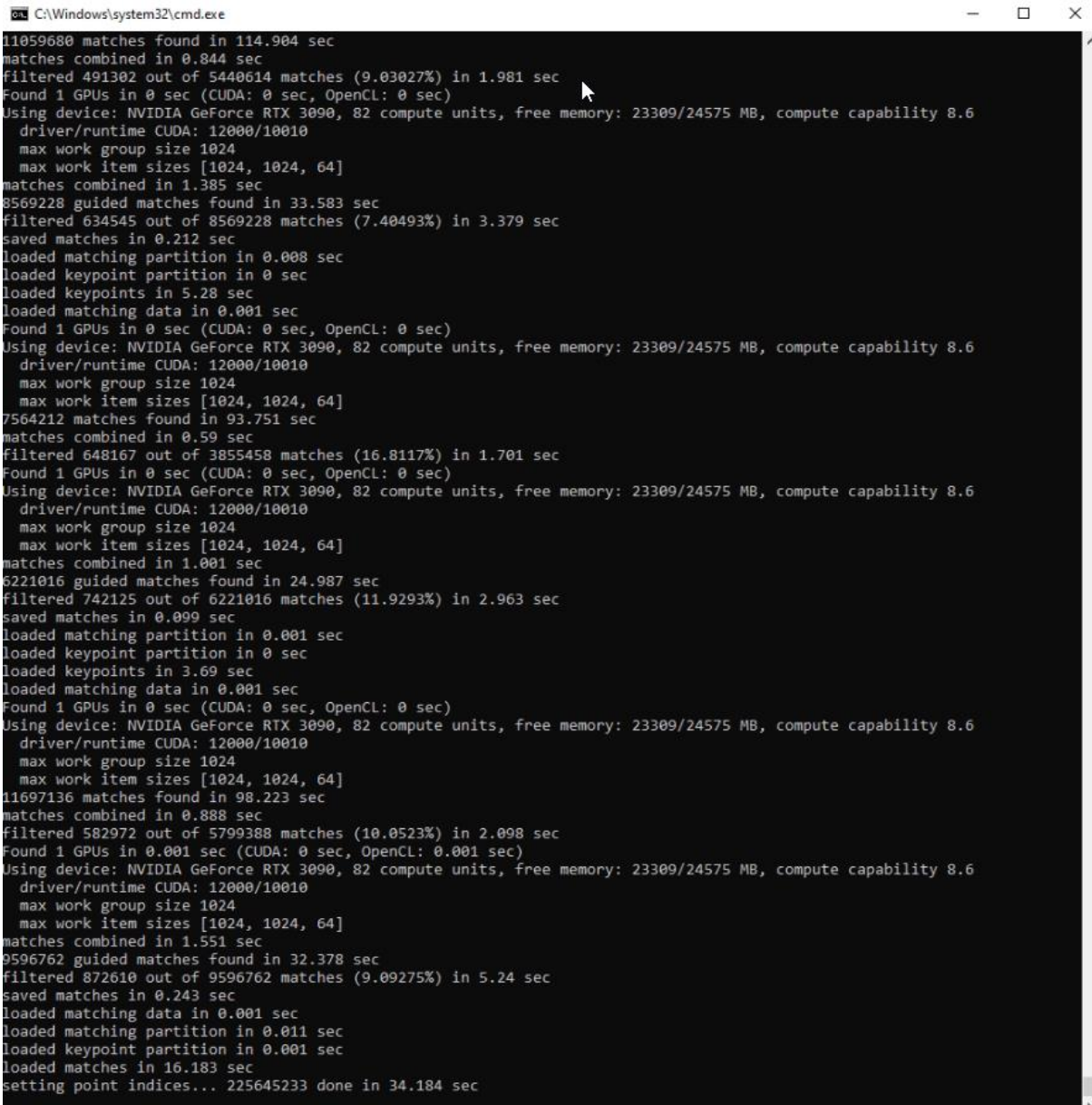
Die reine Berechnungszeit der implementierten GCP-Erkennung durch HCT nahm jeweils mit weniger als 5 Minuten die geringste Berechnungsdauer aller Teilschritte in Anspruch. Der menschliche Bearbeiter würde jeweils über eine Stunde benötigen.

Erfreulicherweise konnte die Implementierung der HCT-Methode zur automatischen Erkennung von GCPs, wie gefordert, ausschließlich mit Open Source Komponenten realisiert werden. Zwar wird innerhalb des Gesamtprojektes derzeit noch Agisoft Metashape eingesetzt, die im Rahmen dieser Arbeit entwickelte Methode kann so jedoch niederschwellig und ohne hohen zusätzlichen Kostenaufwand in einer alternative Softwarelösung, egal ob

proprietär oder Open Source, implementiert werden. Darüber hinaus kann die Methode allen Interessenten frei zur Verfügung gestellt werden.

Das Design der GCPs scheint vorerst gelungen zu sein, was die erreichte Erkennungsrate belegt. Durch die Konstruktion einer Sprühschablone besteht jederzeit die Möglichkeit innerhalb weniger Minuten neue GCPs anzulegen. Diese müssen nicht zwangsläufig stationär sein, sondern können beispielsweise auf einer stabilen Kunststoffplatte und im Notfall auch auf einem DIN-A2 Papier aufgebracht und am Bedarfsort eingemessen werden. Zur Haltbarkeit können momentan nur wenige Rückschlüsse getroffen werden. Die ersten Schnee- und Regenfälle haben zumindest keine sichtbaren Schäden hinterlassen, weiteres wird die Zeit mit sich bringen.

Weitere Verbesserungsmöglichkeiten betreffen vor allem die Nutzerfreundlichkeit (Usability) der Gesamtanwendung. In der hier vorgestellten Variante sind sämtlicher Parameter hardcodiert und für das Projekt BerlinTXL optimiert. Eine Konfigurationsdatei, welche projektspezifische Parameter wie den GCP-Durchmesser, Flughöhen, Upscaling, Erkennungsradien usw. beinhaltet wäre zumindest sinnvoll. Einige Parameter könnten zudem aus den Metadaten der Fotos ausgelesen und zum Teil intelligent bzw. dynamisch berechnet werden. Ergänzend dazu wäre ein grafisches Userinterface (UI) eine Bereicherung für die Nutzererfahrung. Abgesehen von der Parametrierung könnte dort auch eine Fortschrittsanzeige des Auswerteprozesses visualisiert und Datenquellen sowie Backuplaufwerke dynamisch verwaltet werden. Im hier vorgestellten Zustand läuft die Anwendung lediglich in der Windows-Kommandozeile, wie in Abbildung 5-1 dargestellt. Zu guter Letzt wären eine Logdatei für den besseren Rückschluss auf evtl. aufgetretene Fehler sowie ein Report der automatischen GCP-Erkennung eine sinnvolle Ergänzung für den Nutzer.



```
C:\Windows\system32\cmd.exe
11059680 matches found in 114.904 sec
matches combined in 0.844 sec
filtered 491302 out of 5440614 matches (9.03027%) in 1.981 sec
Found 1 GPUs in 0 sec (CUDA: 0 sec, OpenCL: 0 sec)
Using device: NVIDIA GeForce RTX 3090, 82 compute units, free memory: 23309/24575 MB, compute capability 8.6
driver/runtime CUDA: 12000/10010
max work group size 1024
max work item sizes [1024, 1024, 64]
matches combined in 1.385 sec
8569228 guided matches found in 33.583 sec
filtered 634545 out of 8569228 matches (7.40493%) in 3.379 sec
saved matches in 0.212 sec
loaded matching partition in 0.008 sec
loaded keypoint partition in 0 sec
loaded keypoints in 5.28 sec
loaded matching data in 0.001 sec
Found 1 GPUs in 0 sec (CUDA: 0 sec, OpenCL: 0 sec)
Using device: NVIDIA GeForce RTX 3090, 82 compute units, free memory: 23309/24575 MB, compute capability 8.6
driver/runtime CUDA: 12000/10010
max work group size 1024
max work item sizes [1024, 1024, 64]
7564212 matches found in 93.751 sec
matches combined in 0.59 sec
filtered 648167 out of 3855458 matches (16.8117%) in 1.701 sec
Found 1 GPUs in 0 sec (CUDA: 0 sec, OpenCL: 0 sec)
Using device: NVIDIA GeForce RTX 3090, 82 compute units, free memory: 23309/24575 MB, compute capability 8.6
driver/runtime CUDA: 12000/10010
max work group size 1024
max work item sizes [1024, 1024, 64]
matches combined in 1.001 sec
6221016 guided matches found in 24.987 sec
filtered 742125 out of 6221016 matches (11.9293%) in 2.963 sec
saved matches in 0.099 sec
loaded matching partition in 0.001 sec
loaded keypoint partition in 0 sec
loaded keypoints in 3.69 sec
loaded matching data in 0.001 sec
Found 1 GPUs in 0 sec (CUDA: 0 sec, OpenCL: 0 sec)
Using device: NVIDIA GeForce RTX 3090, 82 compute units, free memory: 23309/24575 MB, compute capability 8.6
driver/runtime CUDA: 12000/10010
max work group size 1024
max work item sizes [1024, 1024, 64]
11697136 matches found in 98.223 sec
matches combined in 0.888 sec
filtered 582972 out of 5799388 matches (10.0523%) in 2.098 sec
Found 1 GPUs in 0.001 sec (CUDA: 0 sec, OpenCL: 0.001 sec)
Using device: NVIDIA GeForce RTX 3090, 82 compute units, free memory: 23309/24575 MB, compute capability 8.6
driver/runtime CUDA: 12000/10010
max work group size 1024
max work item sizes [1024, 1024, 64]
matches combined in 1.551 sec
9596762 guided matches found in 32.378 sec
filtered 872610 out of 9596762 matches (9.09275%) in 5.24 sec
saved matches in 0.243 sec
loaded matching data in 0.001 sec
loaded matching partition in 0.011 sec
loaded keypoint partition in 0.001 sec
loaded matches in 16.183 sec
setting point indices... 225645233 done in 34.184 sec
```

Abbildung 5-1: Programmablauf in der Windows Kommandozeile

Zwei weitere große Meilensteine sind aktuell bereits in Arbeit, jedoch noch nicht weit genug fortgeschritten, um in diese Arbeit mit einzufließen. Der erste Meilenstein ist die automatisierte Erkennung und Entfernung von Personen und Fahrzeugen mittels der KI YOLO (Ultralytics 2023) um die Orthomosaik konform zu Datenschutzgrundverordnung³⁶ auch öffentlich zur Verfügung stellen zu können.

³⁶ siehe: <https://dsgvo-gesetz.de/>

Der zweite Meilenstein wird der Umzug weg von der lokalen on-premise Variante in eine Cloudlösung sein. Die Anwendung wird dazu komplett in ein bereits existierendes Kubernetes-Cluster³⁷ eingebunden. Ein Vorteil ist die höhere Rechenleistung und damit verbunden eine schnellere Bearbeitungszeit, welche am Ende wohlmöglich doch noch eine Kreuzbefliegung ermöglichen könnte. Weitere Vorteile sind die höhere Ausfallsicherheit sowie mehr Möglichkeiten für Automatisierung.

Für eine Vollautomatisierung wären dann prinzipiell „nur“ noch zwei Themen zu erörtern: Eine gültige Rechtsgrundlage für vollautonome Flüge sowie das Auswechseln der Akkumulatoren an den Coptern.

³⁷ siehe: <https://www.vmware.com/de/topics/glossary/content/kubernetes-cluster.html>

6 Literaturliste

AG, L. G. (2020). "Leica Zeno FLX100 Smartantenne - Datenblatt." Retrieved 20.02.2023, 2023, from https://leica-geosystems.com/-/media/files/leicageosystems/products/datasheets/leica_zeno_flx100_ds_928858_1020.a_shx?la=de-de&hash=5400BA3798D926D2AA228FFC46776E75.

Agisoft-LLC. (2023). "Agisoft Metashape User Manual Professional Edition, Version 2.0." Retrieved 03.02.2023, 2023, from https://www.agisoft.com/pdf/metashape-pro_2_0_en.pdf.

Allred, B., D. Wishart, L. Martinez, H. Schomberg, S. Mirsky, G. Meyers, J. Elliott and C. Charyton (2018). "Delineation of agricultural drainage pipe patterns using ground penetrating radar integrated with a real-time kinematic global navigation satellite system." Agriculture **8**(11): 167.

Amram, N. (2008). "Hough transform track reconstruction in the cathode strip chambers in ATLAS."

Ang, K. Z. Y., X. Dong, W. Liu, G. Qin, S. Lai, K. Wang, D. Wei, S. Zhang, S. K. Phang and X. Chen (2018). "High-precision multi-UAV teaming for the first outdoor night show in Singapore." Unmanned Systems **6**(01): 39-65.

AVSS. (2022, 11.2022). "PRS-M300 for DJI M300 RTK." Version 2.4. Retrieved 20.02.2023, 2023, from <https://www.avss.co/wp-content/uploads/2022/11/AVSS-PRS-M300-User-Manual-Public-Release-November-2022-2.3.pdf>.

Chen, J., H. Qiang, J. Wu, G. Xu and Z. Wang (2021). "Navigation path extraction for greenhouse cucumber-picking robots using the prediction-point Hough transform." Computers and Electronics in Agriculture **180**: 105911.

Christian Martens, T. G., Tegel Projekt GmbH (2023). Der Einsatz von Drohnen beim Großprojekt Berlin TXL. Forum Wohnen und Stadtwirtschaft, Bundesverband Wohnen und Stadtentwicklung (vhw). **1/2023**: 3.

Colomina, I. and P. Molina (2014). "Unmanned aerial systems for photogrammetry and remote sensing: A review." ISPRS Journal of photogrammetry and remote sensing **92**: 79-97.

DJI. (2020, 2020.05). "MATRICE 300 RTK - Benutzerhandbuch." v1.0. Retrieved 19.02.2023, 2023.

DJI. (2021, 04.2021). "ZENMUSE P1 - Benutzerhandbuch." Retrieved 19.02.2023, 2023, from https://dl.djicdn.com/downloads/Zenmuse_P1/20210618UM/Zenmuse_P1%20_User%20Manual_DE.pdf.

Förstner, W. and B. P. Wrobel (2016). Photogrammetric computer vision, Springer.

Foundation, T. A. S. (2004). "Apache License, Version 2.0." Retrieved 05.03.2023, 2023, from <https://www.apache.org/licenses/LICENSE-2.0>.

GmbH, T. P. (2023) "Berlin TXL – The Urban Tech Republic und das Schumacher Quartier."

- GmbH, T. P. (2023). "Flächenplan." Retrieved 08.02.2023, 2023, from <https://urbantechrepublic.de/der-standort/>.
- Halbrügge, C. and B. Johanning (2015). "Automatisierte Asphaltverdichtung mit schemelgelenkten Tandemwalzen." ATZoffhighway **8**(2): 56-67.
- Hart, P. E. (2009). "How the Hough transform was invented [DSP History]." IEEE Signal Processing Magazine **26**(6): 18-22.
- Harvey, P. (2023). "ExifTool by Phil Harvey." Retrieved 05.03.2023, 2023, from <https://exiftool.org/>.
- Hough, P. V. C. (1962). Method and means for recognizing complex patterns, Google Patents.
- IEC, I. E. C.-. (2013). IEC 60529:1989+AMD1:1999+AMD2:2013 CSV Consolidated version.
- Illingworth, J. and J. Kittler (1988). "A survey of the Hough transform." Computer vision, graphics, and image processing **44**(1): 87-116.
- IONOS-SE. (2022). "Batch-Datei erstellen: So schreiben Sie Ihre eigene Bat-Datei." Was ist eine Batch- bzw. Bat-Datei? Retrieved 30.01.2023, 2023, from <https://www.ionos.at/digitalguide/server/tools/batch-datei-erstellen/>.
- Iqbal, B., W. Iqbal, N. Khan, A. Mahmood and A. Erradi (2020). "Canny edge detection and Hough transform for high resolution video streams using Hadoop and Spark." Cluster Computing **23**(1): 397-408.
- ISO, I. O. f. S.-. (2019). ISO 16684-1:2019.
- James, M. R. and S. Robson (2012). "Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application." Journal of Geophysical Research: Earth Surface **117**(F3).
- James, M. R., S. Robson, S. d'Oleire-Oltmanns and U. Niethammer (2017). "Optimising UAV topographic surveys processed with structure-from-motion: Ground control quality, quantity and bundle adjustment." Geomorphology **280**: 51-66.
- Kaehler, G. B. a. A. (2008). Learning OpenCV, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
- Kaplan, E. D. and C. Hegarty (2017). Understanding GPS/GNSS: principles and applications, Artech house.
- Kersten, T. and M. Lindstaedt (2017). "Photogrammetrie auf Knopfdruck–Auswertung und Resultate UAV-gestützter Bildflugdaten." zfv–Zeitschrift für Geodäsie, Geoinformation und Landmanagement(1): 142.
- Kersten, T., F. Schlömer and H.-J. Przybilla (2020). "Aerotriangulation von UAV-Bilddaten der Zeche Zollern–Die Ergebnisse verschiedener UAV-Systeme und zweier Softwarepakete im Vergleich." DVW e. V.(Hrsg.): UAV: 121-140.
- Kersten, T. P., B. Grahlmann, E. Matthias, M. Lindstaedt and M. Fromhagen (2020). "Bestandsdatenerfassung durch UAV-basierte Bild-flüge am Beispiel einer Straßenkreuzung."

- Kimme, C., D. Ballard and J. Sklansky (1975). "Finding circles by an array of accumulators." Communications of the ACM **18**(2): 120-122.
- Kraus, K. (2007). Photogrammetry, De Gruyter.
- Kraus, K. (2007). Photogrammetry: geometry from images and laser scans, Walter de Gruyter.
- LGB, L. u. G. B.-. (2021). "SAPOS® Faltblatt." Retrieved 05.03.2023, 2023, from https://geobasis-bb.de/sixcms/media.php/9/11_11_21_Faltblatt_LGB_SAPOS_web.pdf.
- Lindstaedt, M. and T. Kersten (2018). "Zur Bedeutung von Passpunkten bei der Aerotriangulation UAV-basierter Bildflüge." DVW e. V.(Hrsg.): UAV: 81-101.
- Luber, t. u. D.-I. F. S. (2023, 08.03.2017). "Definition „Programmierschnittstelle“ Was ist eine API?" Retrieved 05.03.2023, 2023, from <https://www.dev-insider.de/was-ist-eine-api-a-583923/>.
- Mikhail, E. M., J. S. Bethel and J. C. McGlone (2001). Introduction to modern photogrammetry, John Wiley & Sons.
- Montenbruck, O., P. Steigenberger, L. Prange, Z. Deng, Q. Zhao, F. Perosanz, I. Romero, C. Noll, A. Stürze, G. Weber, R. Schmid, K. MacLeod and S. Schaer (2017). "The Multi-GNSS Experiment (MGEX) of the International GNSS Service (IGS) – Achievements, prospects and challenges." Advances in Space Research **59**(7): 1671-1697.
- Nex, F. and F. Remondino (2014). "UAV for 3D mapping applications: A review." Applied Geomatics **6**(1): 1-15.
- OpenCV. (2022, 29.12.2022). "Feature Detection - Function Documentation - HoughCircles()." Retrieved 10.02.2023, 2023, from https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d.
- OpenCV. (2023). "Open Source Computer Vision Library." Retrieved 05.03.2023, 2023, from <https://opencv.org>.
- Pajares, G. (2015). "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)." Photogrammetric Engineering and Remote Sensing **81**(4): 281-329.
- Pan, L., X. Zhang, X. Li, X. Li, C. Lu, J. Liu and Q. Wang (2019). "Satellite availability and point positioning accuracy evaluation on a global scale for integration of GPS, GLONASS, BeiDou and Galileo." Advances in Space Research **63**(9): 2696-2710.
- Peppas, M. V., L. Morelli, J. P. Mills, N. T. Penna and F. Remondino (2022). "HANDCRAFTED AND LEARNING-BASED TIE POINT FEATURES – COMPARISON USING THE EUROSDR RPAS BENCHMARK DATASET." Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. **XLIII-B2-2022**: 1183-1190.
- Peter Boels, E. B.-v. K. (2023). "Markierungsfarbe SOPPEC Tracing Plus." Retrieved 26.01.2023, 2023, from <https://www.boels-surveylaser.de/markierungsfarbe-soppec-tracing-plus/itm/1954>.
- Pirti, A., N. Arslan, B. Deveci, O. Aydin, H. Erkaya and R. G. Hosbas (2009). "Real-time kinematic GPS for cadastral surveying." Survey Review **41**(314): 339-351.

- Project, T. L. I. (2005, 22.04.2005). "BSD License Definition." Retrieved 05.03.2023, 2023, from <http://www.linfo.org/bsdlicense.html>.
- Przybilla, H.-J. and M. Baeumker (2020). RTK and PPK: GNSS-Technologies for direct georeferencing of UAV image flights.
- Przybilla, H.-J., C. Reuber, M. Bäumker and M. Gerke (2015). "Untersuchungen zur Genauigkeitssteigerung von UAV-Bildflügen." Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e. V **24**: 45-54.
- Python-Software-Foundation. (2023). "python." Retrieved 30.01.2023, 2023, from <https://www.python.org/>; <https://docs.python.org/3/>.
- Remondino, F. and S. El-Hakim (2006). "Image-based 3D modelling: a review." The photogrammetric record **21**(115): 269-291.
- Rhody, H. (2005). "Lecture 10: Hough circle transform." Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology.
- Serra, P. L. S., P. H. F. Masotti, M. S. Rocha, D. A. de Andrade, W. M. Torres and R. N. de Mesquita (2020). "Two-phase flow void fraction estimation based on bubble image segmentation using Randomized Hough Transform with Neural Network (RHTN)." Progress in Nuclear Energy **118**: 103133.
- Shibata, T. and W. Frei (1981). Hough transform for target detection in infrared imagery, SPIE.
- Siebert, S. and J. Teizer (2014). "Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system." Automation in Construction **41**: 1-14.
- Štroner, M., R. Urban, J. Seidl, T. Reindl and J. Brouček (2021). "Photogrammetry using UAV-mounted GNSS RTK: Georeferencing strategies without GCPs." Remote Sensing **13**(7): 1336.
- Ultralytics. (2023). "YOLOv5: The friendliest AI architecture you'll ever use." Retrieved 13.03.2023, 2023, from <https://ultralytics.com/yolov5>.
- Vijayarajeswari, R., P. Parthasarathy, S. Vivekanandan and A. A. Basha (2019). "Classification of mammogram for early detection of breast cancer using SVM classifier and Hough transform." Measurement **146**: 800-805.
- Watts, A. C., V. G. Ambrosia and E. A. Hinkley (2012). "Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use." Remote Sensing **4**(6): 1671-1692.
- Wolf, P. R., B. A. Dewitt and B. E. Wilkinson (2014). Elements of Photogrammetry with Applications in GIS, McGraw-Hill Education.
- Wübbena, G., A. Bagge and M. Schmitz (2001). Network-based techniques for RTK applications.
- Yuen, H. K., J. Princen, J. Illingworth and J. Kittler (1990). "Comparative study of Hough Transform methods for circle finding." Image and Vision Computing **8**(1): 71-77.

7 Anhang

7.1 Main.py

```

#import der erforderlichen Module
import Metashape
import sys
import cv2 as cv
import numpy as np
import math
import os
from operator import attrgetter

#Klasse, in der alle Marker-Kamera Kombinationen des Metashape-Projekts gespeichert werden.
class CMarkers():
    def __init__(self, mlabel, cname, cmposx, cmposy, mprojcx=0, mprojcy=0):
        self._mlabel = mlabel      # Markerlabel
        self._cname = cname        # Kamerename (Bild)
        self._cmposx = cmposx      # x-Markerposition auf Bild vor Korrektur (in Bildkoordinaten)
        self._cmposy = cmposy      # y-Markerposition auf Bild vor Korrektur (in Bildkoordinaten)
        self._mprojcx = mprojcx    # berechnete korrigierte x-Markerposition (in Bildkoordinaten)
        self._mprojcy = mprojcy    # berechnete korrigierte y-Markerposition (in Bildkoordinaten)

# Klasse, in der innerhalb der Funktion #opencv die mittels Hough-Circle bestimmten Kreise gespeichert werden
class CVmatches():
    def __init__(self, Px, Py, rad, dist):
        self._Px = Px             # x-Koordinate des bestimmten Kreises
        self._Py = Py             # y-Koordinate des bestimmten Kreises
        self._rad = rad           # Radius des bestimmten Kreises
        self._dist = dist         # berechnete Distanz zwischen Markerposition auf Bild vor Korrektur und Kreismittel
                                  # punkt

# -----#
# Diese Funktion wird von Main.py aufgerufen

def set_markers(homepath, chunk, doc):

    # Erstellung einer leeren Liste
    marker_list = []

    # Iteriert durch alle Marker und Kameras im Chunk
    for marker in chunk.markers:
        for camera in chunk.cameras:

            # Abfrage, ob die Kamera über keine Verortung verfügt. Falls True, dann überspringen
            if not camera.reference.location:
                continue

            # Falls es einen Marker auf der Kamera gibt, dann:...
            if camera.project(marker.position):

                # X- und Y-Bildkoordinate des Markers in getrennten Variablen speichern
                cmposx, cmposy = camera.project(marker.position)

                # den pitch-wert der Kamera in einer Variablen speichern
                pitch = camera.reference.rotation.y

                # Über den Pitch-Wert nach Nadir-Aufnahmen filtern
                if pitch >= -1 and pitch <= 1:

                    # Abfrage, ob die Koordinaten auf in die Ausdehnung der Kamera passen
                    # Falls ja, handelt es sich definitiv um ein Marker-Kamera Paar und wird weiter betrachtet
                    if (0 <= cmposx < camera.sensor.width) and (0 <= cmposy < camera.sensor.height):

                        # Erstellt zu jedem Marker-Kamera Paar ein Objekt in der Klasse CMarkers
                        # Übergibt jedem Objekt jeweils die Attribute Markerlabel, Kamerabel sowie X- und Y
                        #-Bildkoordinate des nicht korrigierten Markers
                        # Speichert jedes Objekt in der Liste #marker_list
                        marker_list.append(CMarkers(marker.label, camera.label, cmposx, cmposy))

    # Iteriert durch jedes Objekt in der marker_list und ruft die Funktion #opencv auf, übergibt das Objekt

```

```

# und den homopath
# Die Funktion gibt die X- und Y-Bildkorrekturkoordinaten an das Objekt zurück
for obj in marker_list:
    opencv(obj, homopath)
    #gis(obj, homopath)

# Iteriert durch alle Marker und Kameras im Chunk
for marker in chunk.markers:
    for camera in chunk.cameras:

        # Abfrage, ob die Kamera über keine Verortung verfügt. Falls True, dann überspringen
        if not camera.reference.location:
            continue

        # Falls es einen Marker auf der Kamera gibt, dann:...
        if camera.project(marker.position):

            # Iteriert durch die Objekte in der Liste
            for newdest in marker_list:

                # gleicht Objektattribute mit den Objektattributen der übergeordneten for-Schleifen ab
                # Wenn Markerlabel und Kamerabel übereinstimmen, dann:
                if newdest._mlabel == marker.label and newdest._cname == camera.label:

                    # Wenn mindestens einer der Korrekturkoordinaten größer als der Default-Wert (0) ist,
                    # dann wurde von #opencv() eine Korrekturcoordinate berechnet
                    if newdest._mprojcx > 0 or newdest._mprojcy > 0:

                        # Korrigieren der Marker-Position auf der Kamera
                        marker.projections[camera] = Metashape.Marker.Projection(Metashape.Vector([newdest._mpro-
jcx,newdest._mprojcy]), True)

                        # Ende der Markerkorrektur, zurück zu Main.py

# -----#

# Diese Funktion wird mit einem Objekt der Klasse CMarkers sowie dem Projektpfad als Parameter von der Funk # tion
set_markers() aufgerufen
# Hier wird jedes Foto, welches einen Marker enthält, mittels Hough Circle Transform nach Kreisen durchsucht
def opencv(obj, homopath):

    # Default Wert für die Variable, in der später die kürzeste Distanz zwischen Kreismittelpunkt und nicht
    # korrigierter Markerposition gespeichert wird
    # Wenn am Ende der Funktion immer noch 99999, dann wurde kein Kreis gefunden
    min_dist = 99999

    if obj._cname is not None:
        # Definieren des Dateinamens des Fotos in Variablen sowie suchen nach dem Foto im Projektverzeichnis
        #über search_files()
        filename = search_jpg(homopath, obj._cname + '.JPG')

        # Öffnen des Fotos in openCV
        src = cv.imread(cv.samples.findFile(filename), cv.IMREAD_COLOR)

        # Überprüfung, ob das Foto nicht geladen werden konnte
        # Falls nein, kommt nur ein Print-Befehl. Könnte perspektivisch in Log-Datei geschrieben werden
        if src is None:
            # Platzhalter für Error-logging
            print("Konnte Foto nicht öffnen!")

        # verschiedene Varianten des Blurrings zu Testzwecken bzw. Customizing
        #gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
        #gray = cv.medianBlur(gray, 5)
        #gray = cv.bilateralFilter(gray,9,250,250)
        #gray = cv.GaussianBlur(gray,(5,5),0)
        #gray = cv.blur(gray,(5,5))
        #gray = cv.adaptiveThreshold(gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY,11,2)

        # Definition der Bounding Box Eckkoordinaten (oben links + unten rechts) zum Clippen des Bildauss-
        #schnittes ausgehend von der Markercoordinate vor Korrektur
        #
        bbX1 = np.uint16(np.around(obj._cmposx)) - 75
        bbY1 = np.uint16(np.around(obj._cmposy)) - 75
        bbX2 = np.uint16(np.around(obj._cmposx)) + 76
        bbY2 = np.uint16(np.around(obj._cmposy)) + 76

```

```

# Mittelpunkt = mittlerer Pixel innerhalb BB = Markerposition vor Korrektur
mX = 76
mY = 76

# Prüfung, ob die Bounding Box partiell außerhalb des Bildbereichs liegt
# Falls ja: Bounding Box an den Bildrand anpassen und Bezug zum ursprünglichen Mittelpunkt (Marker
# position vor Korrektur) beibehalten
if bbX1 <= 0:
    mX = mX + bbX1
    bbX1 = np.uint16(np.around(obj._cmposx)) - np.uint16(np.around(obj._cmposx))
if bbY1 <= 0:
    mY = mY + bbY1
    bbY1 = np.uint16(np.around(obj._cmposy)) - np.uint16(np.around(obj._cmposy))
if bbX2 >= src.shape[1]:
    bbX2 = src.shape[1]
if bbY2 >= src.shape[0]:
    bbY2 = src.shape[0]

# Erzeugen des Bildausschnittes über die Bounding Box
markerarea = src[bbY1:bbY2, bbX1:bbX2]

# Konvertierung des Bildausschnittes in ein Graustufenraster
gray = cv.cvtColor(markerarea, cv.COLOR_BGR2GRAY)

# Upscaling des Bildausschnittes (weitere Stufen können bei Bedarf hinzugeschaltet werden)
gray = cv.pyrUp(gray) #2 # 20 , 30 200
gray = cv.pyrUp(gray) #4 # 40 , 60 400
#gray = cv.pyrUp(gray) #8 # 80 , 120 800
#gray = cv.pyrUp(gray) #16 #160 , 240 1600
#gray = cv.pyrUp(gray) #32 #320 , 480 3200

# Anwenden der Hough Circle Methode auf den Bildausschnitt
circles = cv.HoughCircles (
    gray,
    cv.HOUGH_GRADIENT,
    1,
    604,
    param1 = 50,
    param2 = 30,
    minRadius = 40,
    maxRadius = 65
)

# Prüfen, ob mindestens ein Kreis gefunden wurde
# Falls ja, werden die Berechnungen durchgeführt
# Falls nein, ist die Funktion beendet
if circles is not None:

    # Anlegen einer leeren Liste in der für jeden gefundenen Kreis im Bildausschnitt ein Objekt der
    # Klasse CVmatches gespeichert werden soll
    objdistlist = []

    # Iterieren durch alle gefunden Kreise
    for i in circles[0, :]:

        # Speichern der Mittelpunktkoordinaten des Kreises sowie dessen Radius in separaten Variablen
        Px = i[0]
        Py = i[1]
        rad = i[2]

        # Berechnen der Distanz zwischen dem Mittelpunkt des gefundenen Kreises sowie der Bildkoo-
        # rdinate der Markerposition vor Korrektur
        eDistance = math.dist([Px, Py], [(float(mX * 4)), (float(mY * 4))])

        # Als Objekt der Klasse CVMatches der Liste mit Koordinaten, Radius und berechneter Distanz
        # hinzufügen
        objdistlist.append(CVmatches(Px, Py, rad, eDistance))

    # Prüfen, ob es Einträge in der Liste gibt. Verhindert Abbruch der Routine falls nicht.
    if len(objdistlist) != 0:

        # Filtern nach dem Listeneintrag mit geringster Distanz zur nicht korrigierten Markerposition
        # und speichern in Variablen
        min_dist = min(objdistlist, key=attrgetter('_dist'))

```

```

# Wenn es einen Eintrag in der Liste gab, sollte der eingangs gesetzte Default-Wert '99999' nun
# überschrieben sein und weitere Berechnungen können durchgeführt werden
# Falls der Wert noch '99999' beträgt, ist die Funktion beendet
if min_dist != 99999:

    # Die Distanz-Schwellenwert darf 200 sein, ist die Funktion beendet
    # Filtert mögliche Fehler, Wert sollte nochmals validiert werden
    if min_dist._dist <= 200:

        # Berechnen der korrigierten Markerposition auf Basis der linken oberen Ecke der Bounding Box
        obj._mprojcx = bbX1 + (min_dist._Px / 4)
        obj._mprojcy = bbY1 + (min_dist._Py / 4)

# Visualisierung der Ergebnisse zur besseren Nachvollziehbarkeit und zur Kontrolle

# Abspeichern und Kennzeichnen der Treffer im tmp-Ordner
# OpenCV Funktionen
NearestCenter = tuple(np.uint16(np.around(((min_dist._Px / 4), (min_dist._Py / 4))))))
MarkerCenter0 = tuple(np.uint16(np.around(((mX), (mY))))))
NearestRadius = np.uint16(np.around(min_dist._rad / 4))
cv.circle(markerarea, NearestCenter, 1, (0, 255, 0), 1)
cv.circle(markerarea, MarkerCenter0, 1, (0, 0, 255), 1)
cv.circle(markerarea, NearestCenter, NearestRadius, (0, 255, 0), 1)

NearestCenter = tuple(np.uint16(np.around(((min_dist._Px), (min_dist._Py))))))
MarkerCenter0 = tuple(np.uint16(np.around(((mX * 4), (mY * 4))))))
NearestRadius = np.uint16(np.around(min_dist._rad))
gray2color = cv.cvtColor(gray, cv.COLOR_GRAY2RGB)
cv.circle(gray2color, NearestCenter, 1, (0, 255, 0), 3)
cv.circle(gray2color, MarkerCenter0, 1, (0, 0, 255), 3)
cv.circle(gray2color, NearestCenter, NearestRadius, (0, 255, 0), 1)

# Ausgabe der Bilder im Tempordner
cv.imwrite(homepath + '\\\\' + 'tmp\\' + obj._cname + '_gray' + obj._mlabel + '.JPG', gray2color)
cv.imwrite(homepath + '\\\\' + 'tmp\\' + obj._cname + '_marea' + obj._mlabel + '.JPG', markerarea)

# Rückgabe der korrigierten Markerpositionen
return obj._mprojcx, obj._mprojcy

# gezieltes Recursives durchsuchen des Projektordners nach einer Datei über deren Dateinamen
def search_jpg(folder, filename):
    for root, dirs, files in os.walk(folder):
        for file in files:
            if file == filename:
                return os.path.join(root, file)

```

7.2 Set_Markers_HC.py

```

#import der erforderlichen Module
import Metashape
import sys
import cv2 as cv
import numpy as np
import math
import os
from operator import attrgetter

#Klasse, in der alle Marker-Kamera Kombinationen des Metashape-Projekts gespeichert werden.
class CMarkers():
    def __init__(self, mlabel, cname, cmposx, cmposy, mprojcx=0, mprojcy=0):
        self._mlabel = mlabel # Markerlabel
        self._cname = cname # Kamerename (Bild)
        self._cmposx = cmposx # x-Markerposition auf Bild vor Korrektur (in Bildkoordinaten)
        self._cmposy = cmposy # y-Markerposition auf Bild vor Korrektur (in Bildkoordinaten)
        self._mprojcx = mprojcx # berechnete korrigierte x-Markerposition (in Bildkoordinaten)
        self._mprojcy = mprojcy # berechnete korrigierte y-Markerposition (in Bildkoordinaten)

# Klasse, in der innerhalb der Funktion #opencv die mittels Hough-Circle bestimmten Kreise gespei

```

```

# chert werden
class CVmatches():
    def __init__(self, Px, Py, rad, dist):
        self._Px = Px      # x-Koordinate des bestimmten Kreises
        self._Py = Py      # y-Koordinate des bestimmten Kreises
        self._rad = rad    # Radius des bestimmten Kreises
        self._dist = dist  # berechnete Distanz zwischen Markerposition auf Bild vor Korrektur und Kreismittelpunkt

# -----#
# Diese Funktion wird von Main.py aufgerufen

def set_markers(homepath, chunk, doc):

    # Erstellung einer leeren Liste
    marker_list = []

    # Iteriert durch alle Marker und Kameras im Chunk
    for marker in chunk.markers:
        for camera in chunk.cameras:

            # Abfrage, ob die Kamera über keine Verortung verfügt. Falls True, dann überspringen
            if not camera.reference.location:
                continue

            # Falls es einen Marker auf der Kamera gibt, dann:...
            if camera.project(marker.position):

                # X- und Y-Bildkoordinate des Markers in getrennten Variablen speichern
                cmposx, cmposy = camera.project(marker.position)

                # den pitch-wert der Kamera in einer Variablen speichern
                pitch = camera.reference.rotation.y

                # Über den Pitch-Wert nach Nadir-Aufnahmen filtern
                if pitch >= -1 and pitch <= 1:

                    # Abfrage, ob die Koordinaten auf in die Ausdehnung der Kamera passen
                    # Falls ja, handelt es sich definitiv um ein Marker-Kamera Paar und wird weiter
                    # betrachtet
                    if (0 <= cmposx < camera.sensor.width) and (0 <= cmposy < camera.sensor.height):

                        # Erstellt zu jedem Marker-Kamera Paar ein Objekt in der Klasse CMarkers
                        # Übergibt jedem Objekt jeweils die Attribute Markerlabel,
                        # Kameralabel sowie X- und Y-Bildkoordinate des nicht korrigierten Markers
                        # Speichert jedes Objekt in der Liste #marker_list
                        marker_list.append(CMarkers(marker.label, camera.label, cmposx, cmposy))

            # Iteriert durch jedes Objekt in der marker_list und ruft die Funktion #opencv auf, übergibt das Objekt
            # und den homepath
            # Die Funktion gibt die X- und Y-Bildkorrekturkoordinaten an das Objekt zurück
            for obj in marker_list:
                opencv(obj, homepath)
                #gis(obj, homepath)

            # Iteriert durch alle Marker und Kameras im Chunk
            for marker in chunk.markers:
                for camera in chunk.cameras:

                    # Abfrage, ob die Kamera über keine Verortung verfügt. Falls True, dann überspringen
                    if not camera.reference.location:
                        continue

                    # Falls es einen Marker auf der Kamera gibt, dann:...
                    if camera.project(marker.position):

                        # Iteriert durch die Objekte in der Liste
                        for newest in marker_list:

                            # gleicht Objektattribute mit den Objektattributen der übergeordneten for-Schleifen ab
                            # Wenn Markerlabel und Kameralabel übereinstimmen, dann:
                            if newest._mlabel == marker.label and newest._cname == camera.label:

                                # Wenn mindestens einer der Korrekturkoordinaten größer als der Default-Wert (0) ist,
                                # dann wurde von #opencv() eine Korrekturkoordinate berechnet
                                if newest._mprojcx > 0 or newest._mprojcy > 0:

```

```
        # Korrigieren der Marker-Position auf der Kamera
        marker.projections[camera] = Metashape.Marker.Projection(Metashape.Vector([newdest._mprojc-
jcx,newdest._mprojcy]), True)

        # Ende der Markerkorrektur, zurück zu Main.py

# ----- #

# Diese Funktion wird mit einem Objekt der Klasse CMarkers sowie dem Projektpfad als Parameter von der Funktion
# set_markers() aufgerufen
# Hier wird jedes Foto, welches einen Marker enthält, mittels Hough Circle Transform nach Kreisen durchsucht
def opencv(obj, homepath):

    # Default Wert für die Variable, in der später die kürzeste Distanz zwischen Kreismittelpunkt und nicht
    # korrigierter Markerposition gespeichert wird
    # Wenn am Ende der Funktion immer noch 99999, dann wurde kein Kreis gefunden
    min_dist = 99999

    if obj._cname is not None:
        # Definieren des Dateinamens des Fotos in Variablen sowie suchen nach dem Foto im Projektverzeichnis über
        # search_files()
        filename = search_jpg(homepath, obj._cname + '.JPG')

        # Öffnen des Fotos in openCV
        src = cv.imread(cv.samples.findFile(filename), cv.IMREAD_COLOR)

        # Überprüfung, ob das Foto nicht geladen werden konnte
        # Falls nein, kommt nur ein Print-Befehl. Könnte perspektivisch in Log-Datei geschrieben werden
        if src is None:
            # Platzhalter für Error-logging
            print("Konnte Foto nicht öffnen!")

        # verschiedene Varianten des Blurrings zu Testzwecken bzw. Customizing
        #gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
        #gray = cv.medianBlur(gray, 5)
        #gray = cv.bilateralFilter(gray,9,250,250)
        #gray = cv.GaussianBlur(gray,(5,5),0)
        #gray = cv.blur(gray,(5,5))
        #gray = cv.adaptiveThreshold(gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY,11,2)

        # Definition der Bounding Box Eckkoordinaten (oben links + unten rechts) zum Clippen des
        # Bildausschnittes ausgehend von der Markerkoordinate vor Korrektur

        bbX1 = np.uint16(np.around(obj._cmposx)) - 75
        bbY1 = np.uint16(np.around(obj._cmposy)) - 75
        bbX2 = np.uint16(np.around(obj._cmposx)) + 76
        bbY2 = np.uint16(np.around(obj._cmposy)) + 76

        # Mittelpunkt = mittlerer Pixel innerhalb BB = Markerposition vor Korrektur
        mX = 76
        mY = 76

        # Prüfung, ob die Bounding Box partiell außerhalb des Bildbereichs liegt
        # Falls ja: Bounding Box an den Bildrand anpassen und Bezug zum ursprünglichen Mittelpunkt
        # (Markerposition vor Korrektur) beibehalten
        if bbX1 <= 0:
            mX = mX + bbX1
            bbX1 = np.uint16(np.around(obj._cmposx)) - np.uint16(np.around(obj._cmposx))
        if bbY1 <= 0:
            mY = mY + bbY1
            bbY1 = np.uint16(np.around(obj._cmposy)) - np.uint16(np.around(obj._cmposy))
        if bbX2 >= src.shape[1]:
            bbX2 = src.shape[1]
        if bbY2 >= src.shape[0]:
            bbY2 = src.shape[0]

        # Erzeugen des Bildausschnittes über die Bounding Box
        markerarea = src[bbY1:bbY2, bbX1:bbX2]

        # Konvertierung des Bildausschnittes in ein Graustufenraster
        gray = cv.cvtColor(markerarea, cv.COLOR_BGR2GRAY)

        # Upscaling des Bildausschnittes (weitere Stufen können bei Bedarf hinzugeschaltet werden)
        gray = cv.pyrUp(gray) #2 # 20 , 30 200
```



```

gray = cv.pyrUp(gray) #4 # 40 , 60 400
#gray = cv.pyrUp(gray) #8 # 80 , 120 800
#gray = cv.pyrUp(gray) #16 #160 , 240 1600
#gray = cv.pyrUp(gray) #32 #320 , 480 3200

# Anwenden der Hough Circle Methode auf den Bildausschnitt
circles = cv.HoughCircles (
    gray,
    cv.HOUGH_GRADIENT,
    1,
    604,
    param1 = 50,
    param2 = 30,
    minRadius = 40,
    maxRadius = 65
)

# Prüfen, ob mindestens ein Kreis gefunden wurde
# Falls ja, werden die Berechnungen durchgeführt
# Falls nein, ist die Funktion beendet
if circles is not None:

    # Anlegen einer leeren Liste in der für jeden gefundenen Kreis im Bildausschnitt ein Objekt der Klasse
    # CVMatches gespeichert werden soll
    objdistlist = []

    # Iterieren durch alle gefunden Kreise
    for i in circles[0, :]:

        # Speichern der Mittelpunktkoordinaten des Kreises sowie dessen Radius in separaten Variablen
        Px = i[0]
        Py = i[1]
        rad = i[2]

        # Berechnen der Distanz zwischen dem Mittelpunkt des gefundenen Kreises sowie der Bildkoordinate der
        # Markerposition vor Korrektur
        eDistance = math.dist([Px, Py],[float(mX * 4), float(mY * 4)])

        # Als Objekt der Klasse CVMatches der Liste mit Koordinaten, Radius und berechneter Distanz hinzufügen
        objdistlist.append(CVMatches(Px, Py, rad, eDistance))

    # Prüfen, ob es Einträge in der Liste gibt. Verhindert Abbruch der Routine falls nicht.
    if len(objdistlist) != 0:

        # Filtern nach dem Listeneintrag mit geringster Distanz zur nicht korrigierten
        # Markerposition und speichern in Variablen
        min_dist = min(objdistlist,key=attrgetter('_dist'))

        # Wenn es einen Eintrag in der Liste gab, sollte der eingangs gesetzte
        # Default-Wert '99999' nun überschrieben sein und weitere Berechnungen
        # können durchgeführt werden
        # Falls der Wert noch '99999' beträgt, ist die Funktion beendet
        if min_dist != 99999:

            # Die Distanz-Schwellenwert darf 200 sein, ist die Funktion beendet
            # Filtert mögliche Fehler, Wert sollte nochmals validiert werden
            if min_dist._dist <= 200:

                # Berechnen der korrigierten Markerposition auf Basis der linken oberen Ecke der Bounding Box
                obj._mprojcx = bbX1 + (min_dist._Px / 4)
                obj._mprojcy = bbY1 + (min_dist._Py / 4)

            # Visualisierung der Ergebnisse zur besseren Nachvollziehbarkeit und zur Kontrolle

            # Abspeichern und Kennzeichnen der Treffer im tmp-Ordner
            # OpenCV Funktionen
            NearestCenter = tuple(np.uint16(np.around(((min_dist._Px / 4), (min_dist._Py / 4)))))
            MarkerCenter0 = tuple(np.uint16(np.around(((mX), (mY)))))
            NearestRadius = np.uint16(np.around(min_dist._rad / 4))
            cv.circle(markerarea, NearestCenter, 1, (0, 255, 0), 1)
            cv.circle(markerarea, MarkerCenter0, 1, (0, 0, 255), 1)
            cv.circle(markerarea, NearestCenter, NearestRadius, (0, 255, 0), 1)

            NearestCenter = tuple(np.uint16(np.around(((min_dist._Px), (min_dist._Py)))))
            MarkerCenter0 = tuple(np.uint16(np.around(((mX * 4), (mY * 4)))))
            NearestRadius = np.uint16(np.around(min_dist._rad))

```

```
gray2color = cv.cvtColor(gray, cv.COLOR_GRAY2RGB)
cv.circle(gray2color, NearestCenter, 1, (0, 255, 0), 3)
cv.circle(gray2color, MarkerCenter0, 1, (0, 0, 255), 3)
cv.circle(gray2color, NearestCenter, NearestRadius, (0, 255, 0), 1)

# Ausgabe der Bilder im Tempordner
cv.imwrite(homepath + '\\\\' + 'tmp\\\\' + obj._cname + '_gray' + obj._mlabel + '.JPG', gray2color)
cv.imwrite(homepath + '\\\\' + 'tmp\\\\' + obj._cname + '_marea' + obj._mlabel + '.JPG', markerarea)

# Rückgabe der korrigierten Markerpositionen
return obj._mprojcx, obj._mprojcy

# gezieltes Recursives durchsuchen des Projektordners nach einer Datei über deren Dateinamen
def search_jpg(folder, filename):
    for root, dirs, files in os.walk(folder):
        for file in files:
            if file == filename:
                return os.path.join(root, file)
```

7.3 Beispiel XMP Metadaten

```
ExifTool Version Number      : 12.55
File Name                    : DJI_20230223105455_0060.JPG
Directory                   : C:/Auswerten/20230223_Gesamtareal/DJI_202302231048_001_UAV120230117
File Size                   : 14 MB
File Modification Date/Time  : 2023:02:23 10:54:54+01:00
File Access Date/Time       : 2023:03:05 00:05:09+01:00
File Creation Date/Time     : 2023:02:23 15:49:54+01:00
File Permissions            : -rw-rw-rw-
File Type                   : JPEG
File Type Extension         : jpg
MIME Type                   : image/jpeg
Exif Byte Order             : Little-endian (Intel, II)
Image Description           : default
Make                       : DJI
Camera Model Name          : ZennuseP1
Orientation                 : Horizontal (normal)
X Resolution                : 72
Y Resolution                : 72
Resolution Unit             : inches
Software                   : 04.00.02.05
Modify Date                 : 2023:02:23 10:54:55
Y Cb Cr Positioning        : Co-sited
Exposure Time              : 1/1000
F Number                   : 5.0
Exposure Program           : Program AE
ISO                        : 800
Sensitivity Type           : Recommended Exposure Index
Exif Version               : 0230
Date/Time Original         : 2023:02:23 10:54:55
Create Date                : 2023:02:23 10:54:55
Components Configuration   : Y, Cb, Cr, -
Shutter Speed Value        : 1/1000
Aperture Value             : 5.0
Exposure Compensation      : 0
Max Aperture Value        : 2.8
Metering Mode              : Average
Light Source               : Fluorescent
Flash                     : No Flash
Focal Length               : 35.0 mm
AE Debug Info              : (Binary data 4096 bytes, use -b option to extract)
AE Histogram Info         : (Binary data 4096 bytes, use -b option to extract)
AE Local Histogram        : (Binary data 2048 bytes, use -b option to extract)
AE Live View Histogram Info : (Binary data 4096 bytes, use -b option to extract)
AE Live View Local Histogram : (Binary data 2048 bytes, use -b option to extract)
AWB Debug Info            : (Binary data 4096 bytes, use -b option to extract)
AF Debug Info             : (Binary data 256 bytes, use -b option to extract)
Histogram                 : disable,
Xidiri                    : (Binary data 512 bytes, use -b option to extract)
Gimbal Degree             : -993,-899,1800
```

```
Flight Degree : 728,-67,23
ADJ Debug Info : (Binary data 1024 bytes, use -b option to extract)
Sensor ID : 3XM5K6G001FMDU
Flight Speed : 29,95,0
Hyperlaps Debug Info : (Binary data 8 bytes, use -b option to extract)
Flashpix Version : 0100
Color Space : sRGB
Exif Image Width : 8192
Exif Image Height : 5460
Interoperability Index : R98 - DCF basic file (sRGB)
Interoperability Version : 0100
File Source : Digital Camera
Scene Type : Directly photographed
Exposure Mode : Auto
White Balance : Auto
Digital Zoom Ratio : 1
Focal Length In 35mm Format : 35 mm
Scene Capture Type : Standard
Gain Control : None
Contrast : Normal
Saturation : Normal
Sharpness : Normal
Device Setting Description : (Binary data 4 bytes, use -b option to extract)
Serial Number : 3XM5K6G001FMDU
Lens Info : 35mm f/2.8
Lens Serial Number : 01KY117G010P
GPS Version ID : 2.3.0.0
GPS Latitude Ref : North
GPS Longitude Ref : East
GPS Altitude Ref : Above Sea Level
GPS Status : Measurement Active
GPS Map Datum : WGS-84
XP Comment : 0.9.142
XP Keywords : single
Compression : JPEG
Thumbnail Offset : 25006
Thumbnail Length : 9634
About : DJI Meta Data
Format : image/jpeg
Gps Status : RTK
Altitude Type : RtkAlt
Absolute Altitude : +194.736
Relative Altitude : +120.095
Gimbal Roll Degree : +180.00
Gimbal Yaw Degree : -99.30
Gimbal Pitch Degree : -89.90
Flight Roll Degree : +2.30
Flight Yaw Degree : +72.80
Flight Pitch Degree : -6.70
Flight X Speed : 2.9
Flight Y Speed : 9.5
```

```
Flight Z Speed           : 0.0
Cam Reverse              : 0
Gimbal Reverse           : 0
Self Data                :
Rtk Flag                 : 50
Rtk Std Lon              : 0.01228
Rtk Std Lat               : 0.01843
Rtk Std Hgt              : 0.02359
Rtk Diff Age             : 1.80000
Surveying Mode           : 1
UTC At Exposure          : 2023:02:23 09:55:13.506045
Shutter Type             : Mechanical
Shutter Count            : 31167
Camera Serial Number     : 3XM5K6G001FMDU
Drone Model              : Matrice 300 RTK
Drone Serial Number      : 1ZN3K8P001P00U
Version                  : 7.0
Has Settings             : False
Has Crop                 : False
Already Applied          : False
MPF Version              : 0100
Number Of Images         : 2
MP Image Flags           : Dependent child image
MP Image Format           : JPEG
MP Image Type            : Large Thumbnail (VGA equivalent)
MP Image Length          : 722229
MP Image Start           : 13257906
Dependent Image 1 Entry Number : 0
Dependent Image 2 Entry Number : 0
Image UID List           : (Binary data 66 bytes, use -b option to extract)
Total Frames             : 1
Image Width              : 8192
Image Height             : 5460
Encoding Process         : Baseline DCT, Huffman coding
Bits Per Sample          : 8
Color Components         : 3
Y Cb Cr Sub Sampling    : YCbCr4:2:0 (2 2)
Aperture                 : 5.0
Image Size               : 8192x5460
Megapixels               : 44.7
Scale Factor To 35 mm Equivalent: 1.0
Shutter Speed            : 1/1000
Thumbnail Image          : (Binary data 9634 bytes, use -b option to extract)
GPS Altitude             : 194.7 m Above Sea Level
GPS Latitude              : 52 deg 33' 37.43" N
GPS Longitude            : 13 deg 19' 7.79" E
Preview Image            : (Binary data 722229 bytes, use -b option to extract)
Circle Of Confusion     : 0.030 mm
Field Of View            : 54.4 deg
Focal Length             : 35.0 mm (35 mm equivalent: 35.0 mm)
GPS Position              : 52 deg 33' 37.43" N, 13 deg 19' 7.79" E
```

Hyperfocal Distance : 8.15 m
Light Value : 11.6

7.4 Ground Control Points

n;x;y;z

1;384311.244;5824113.010;34.398
2;383481.551;5824445.184;33.544
3;383305.787;5824094.896;33.628
4;383936.839;5823901.213;33.896
5;383558.036;5823915.800;33.933
6;382990.382;5824057.175;32.335
7;382439.861;5824246.613;33.062
8;381829.165;5824214.685;32.799
9;382446.445;5824739.425;33.192
10;383388.976;5824903.833;34.474
11;384296.137;5825016.671;35.882
12;385381.404;5825074.699;36.017
13;385967.388;5824891.640;36.296
14;385394.788;5824561.297;35.820
15;384831.031;5824386.024;34.776
16;384595.299;5823953.920;33.918
17;384838.260;5823802.441;34.331
18;384197.376;5823646.116;34.555
19;383796.697;5824279.877;33.952
20;384905.303;5824843.444;35.497
21;383871.769;5824702.820;35.077
22;382881.711;5824567.240;31.632
23;382124.280;5824448.986;33.000

7.5 Checkpoints

n;x;y;z

CP1;381861.916000000026543;5824503.780000000260770;33.130000000000003

CP2;383809.625999999989290;5824490.019999999552965;34.058000000000000

CP3;383856.45199999990221;5824969.729000000283122;35.387999999999998

CP4;384185.293000000005122;5823654.213999999687076;34.347999999999999

CP5;385397.864000000001397;5825176.953999999910593;35.439000000000000

CP6;384936.161000000021886;5824409.792000000365078;35.186000000000000

CP7;384597.126999999978580;5823957.358000000007451;33.802000000000000

CP8;383856.319000000017695;5824019.368999999947846;33.841999999999999

CP9;382538.985999999975320;5824164.209999999962747;32.621000000000002

7.6 verwendete Software

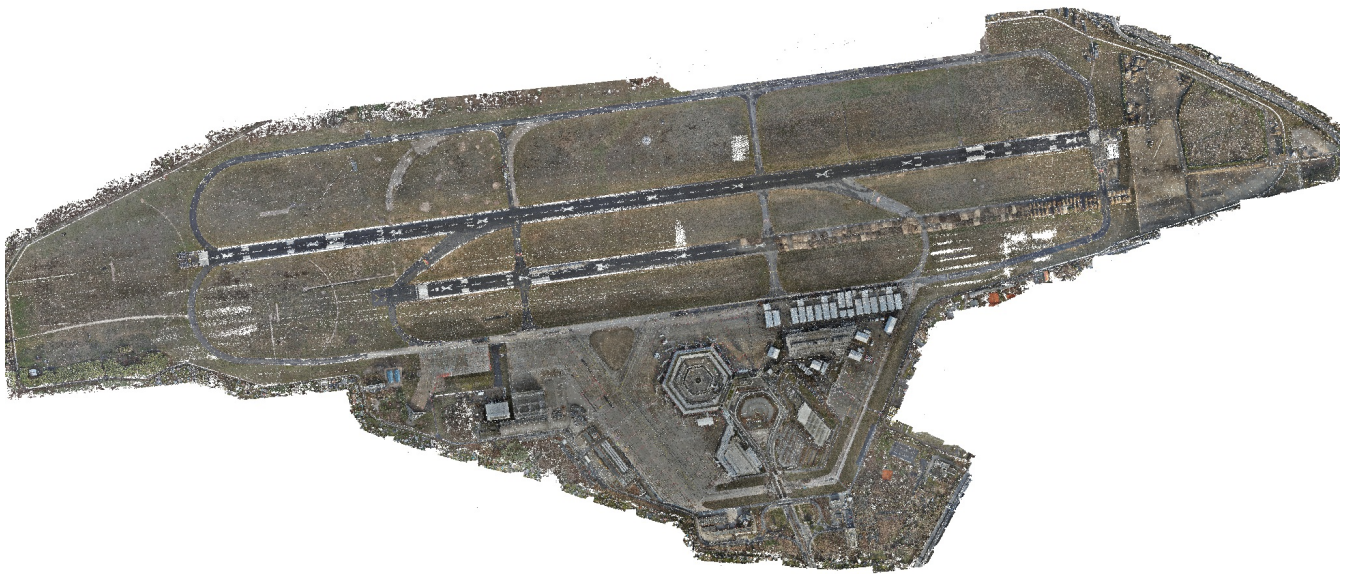
- Agisoft Metashape 3.0
- ArcGIS Pro 3.0
- Greenshot 1.2.10
- MS Office 365
- Endnote 20.5
- Windows 11
- Python 3.8 + Module aus Kapitel 2.2.4.4

7.7 Metashape Protokolle

Agisoft Metashape

Processing Report

10 March 2023



Survey Data

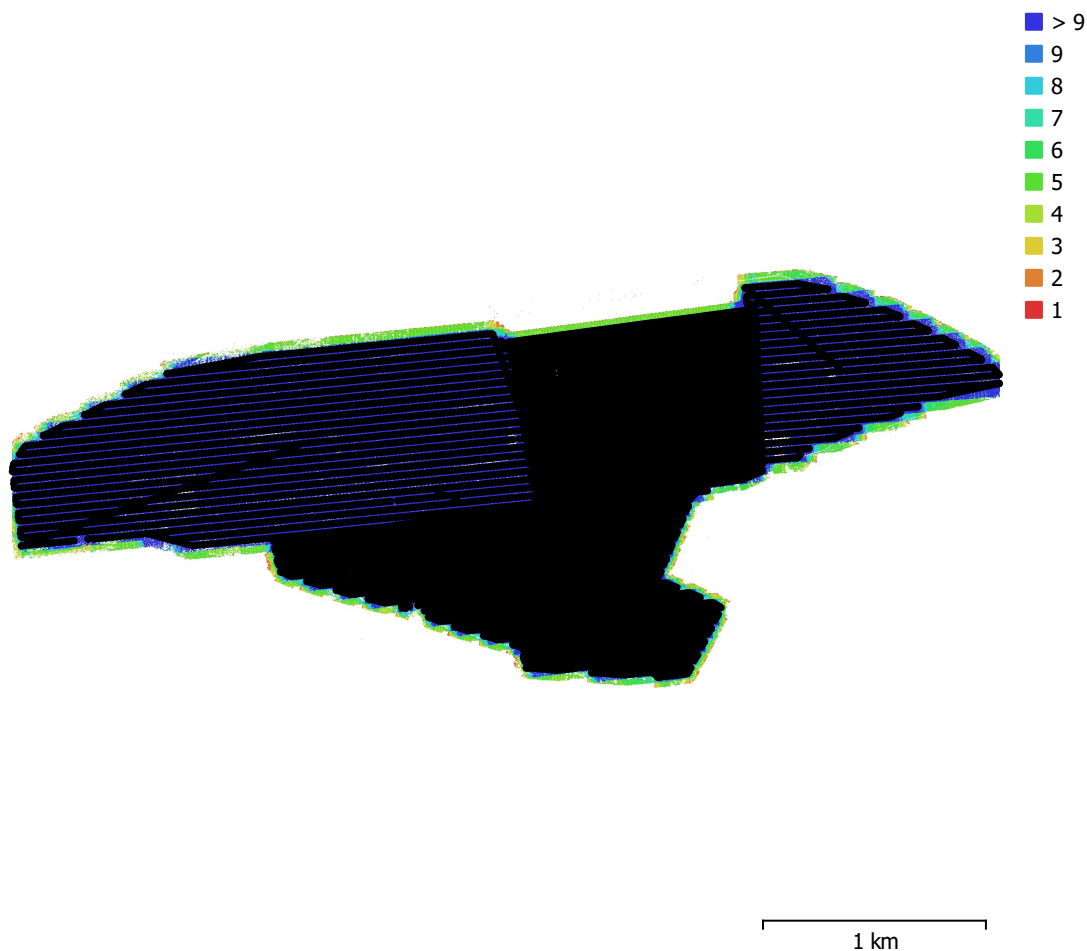


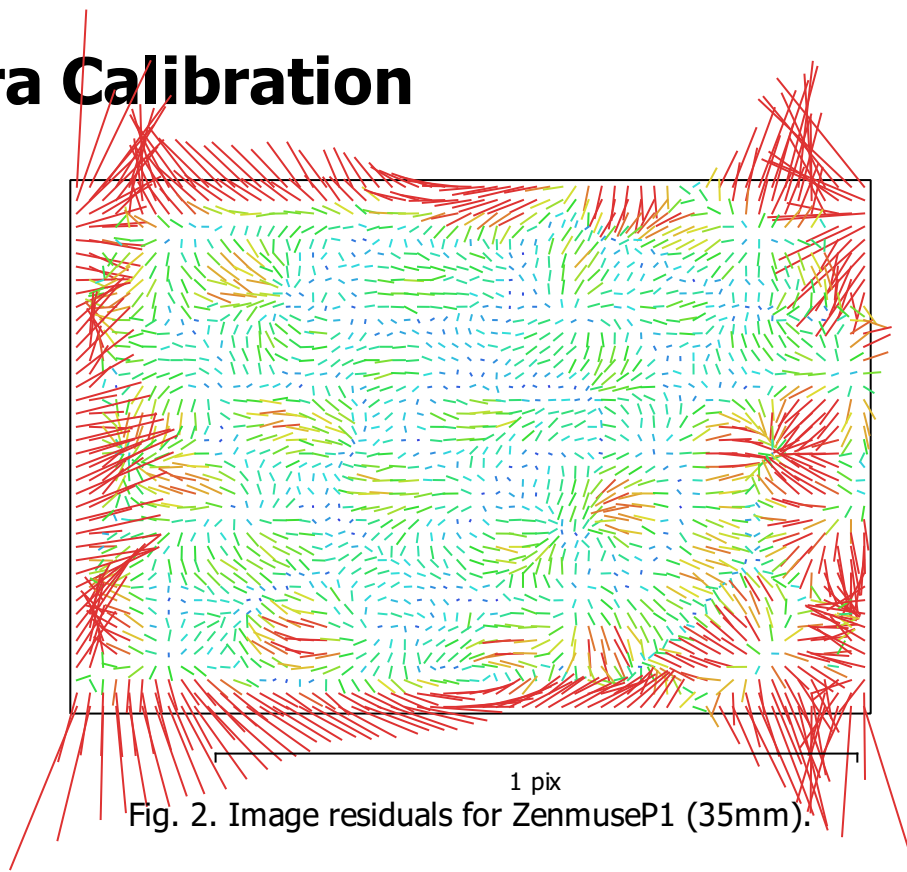
Fig. 1. Camera locations and image overlap.

Number of images:	10,064	Camera stations:	10,054
Flying altitude:	92.7 m	Tie points:	9,169,276
Ground resolution:	1.13 cm/pix	Projections:	79,257,630
Coverage area:	4.35 km ²	Reprojection error:	0.68 pix

Camera Model	Resolution	Focal Length	Pixel Size	Precalibrated
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No

Table 1. Cameras.

Camera Calibration



ZenmuseP1 (35mm)

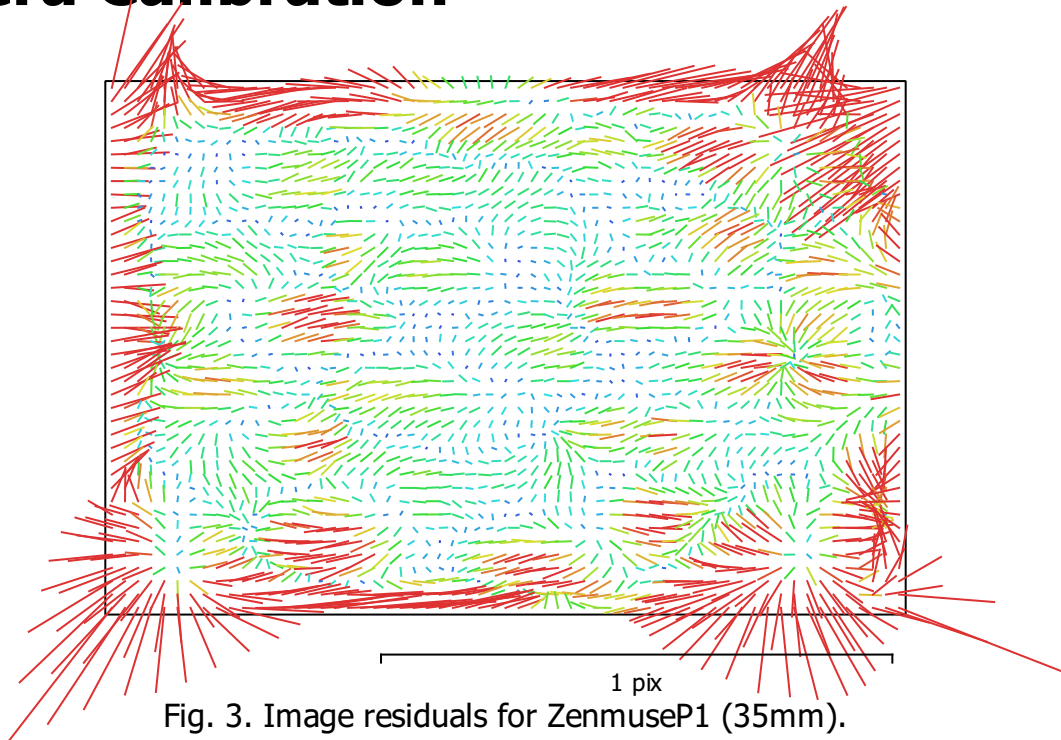
4315 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8199.33	0.044	1.00	-0.05	-0.01	-0.98	0.96	-0.94	0.91	-0.04	-0.02
Cx	-27.9677	0.019		1.00	0.02	0.05	-0.06	0.06	-0.06	0.98	0.00
Cy	27.1125	0.019			1.00	0.02	-0.02	0.02	-0.01	0.01	0.99
K1	-0.0495387	9.6e-05				1.00	-1.00	0.98	-0.96	0.04	0.02
K2	0.148205	0.00063					1.00	-0.99	0.98	-0.05	-0.02
K3	-0.464436	0.0018						1.00	-0.99	0.05	0.02
K4	0.379889	0.0018							1.00	-0.05	-0.02
P1	-0.0025101	1.1e-06								1.00	-0.00
P2	0.00144497	1.1e-06									1.00

Table 2. Calibration coefficients and correlation matrix.

Camera Calibration



ZenmuseP1 (35mm)

5749 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8197.9	0.038	1.00	-0.01	-0.01	-0.97	0.96	-0.93	0.90	-0.01	-0.02
Cx	-9.9874	0.017		1.00	0.01	0.01	-0.01	0.01	-0.01	0.98	-0.00
Cy	39.6876	0.017			1.00	0.02	-0.02	0.01	-0.01	0.00	0.99
K1	-0.0428332	8.4e-05				1.00	-0.99	0.98	-0.95	0.01	0.03
K2	0.116851	0.00055					1.00	-0.99	0.98	-0.01	-0.03
K3	-0.381426	0.0016						1.00	-0.99	0.01	0.02
K4	0.297893	0.0016							1.00	-0.00	-0.02
P1	-0.000922268	9.3e-07								1.00	-0.01
P2	0.00233483	9.7e-07									1.00

Table 3. Calibration coefficients and correlation matrix.

Camera Locations

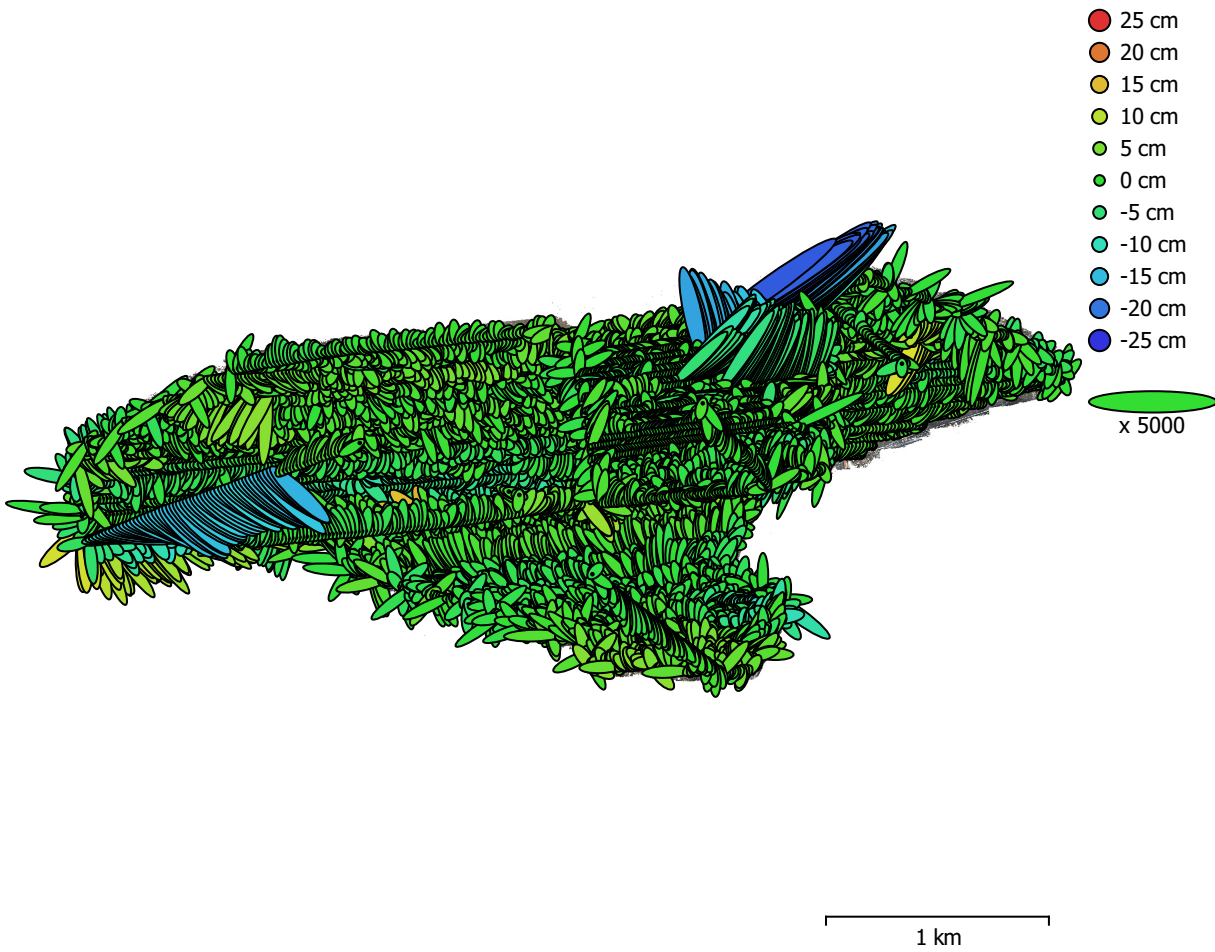


Fig. 4. Camera locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated camera locations are marked with a black dot.

X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total error (cm)
1.32117	1.5839	3.2366	2.06258	3.83794

Table 4. Average camera location error.

X - Easting, Y - Northing, Z - Altitude.

Ground Control Points

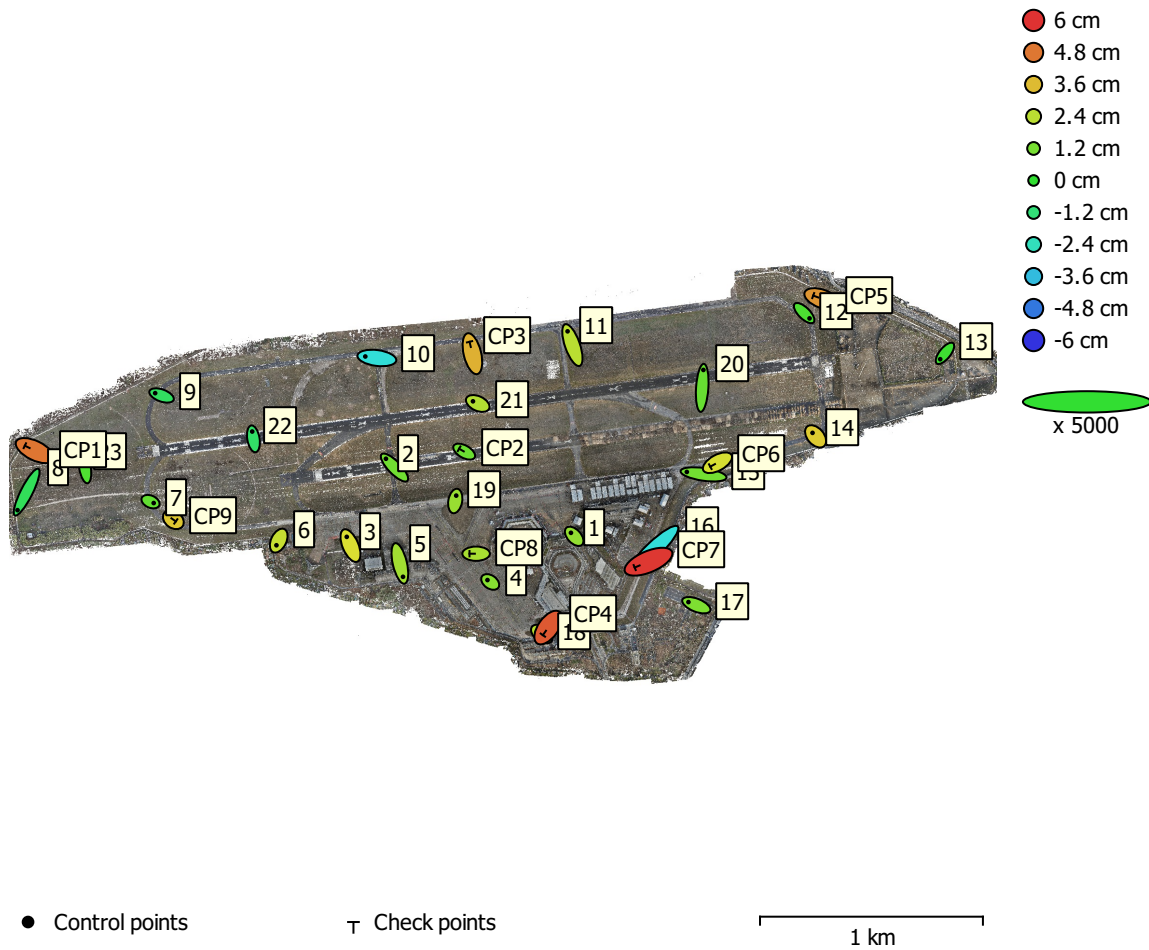


Fig. 5. GCP locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated GCP locations are marked with a dot or crossing.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
23	1.30339	1.5986	1.85719	2.06261	2.77552

Table 5. Control points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
9	1.3872	1.02711	4.0004	1.72606	4.35689

Table 6. Check points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
1	-0.578819	0.624547	1.67793	1.88163	0.887 (12)
2	-1.52801	1.57739	1.09025	2.45186	0.945 (16)
3	-0.666656	1.56552	2.94385	3.40023	0.852 (22)
4	-0.490926	0.339998	1.46119	1.57851	0.876 (15)
5	0.563832	-2.39918	1.94116	3.13721	1.044 (11)
6	-0.409209	-0.833578	2.59121	2.75257	0.766 (23)
7	0.571056	-0.248717	0.841896	1.04726	0.675 (15)
8	-1.63259	-3.2338	-0.678742	3.68558	0.961 (12)
9	-1.15015	0.416218	-0.79986	1.46146	0.743 (15)
10	-2.07263	0.268564	-2.90975	3.58253	1.097 (15)
11	-0.866875	2.47116	2.37804	3.5374	1.223 (10)
12	0.970094	-0.950475	0.443743	1.42877	0.731 (15)
13	-0.852706	-1.05158	0.207103	1.3696	0.726 (14)
14	-0.549326	0.626236	3.32793	3.4306	1.578 (12)
15	-2.9969	0.435214	1.12369	3.23009	0.974 (16)
16	-3.39111	-3.25961	-2.90415	5.528	1.004 (14)
17	-1.41499	0.560294	1.32585	2.01842	0.739 (15)
18	0.688783	-0.514881	2.29775	2.4534	0.735 (12)
19	0.174484	0.967453	1.77947	2.03296	0.832 (19)
20	0.211992	3.21114	0.918988	3.34677	0.995 (15)
21	-0.855924	0.387975	2.22534	2.41563	0.939 (19)
22	-0.218999	1.3091	-0.914521	1.61185	0.765 (20)
23	-0.295671	1.76165	0.513102	1.85852	0.840 (14)
Total	1.30339	1.5986	1.85719	2.77552	0.911

Table 7. Control points.
X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP1	-1.80904	0.882896	4.80869	5.21302	0.177 (16)
CP2	-0.913072	0.512989	0.98764	1.43955	0.306 (15)
CP3	-0.514556	2.13578	3.80847	4.39668	0.326 (13)

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP4	-0.966625	-1.36125	5.32454	5.58015	0.258 (15)
CP5	-1.60898	0.593652	4.26774	4.59944	0.336 (21)
CP6	-1.29253	-0.701761	2.86677	3.22203	0.260 (13)
CP7	-2.48067	-1.04727	5.90684	6.49163	0.497 (16)
CP8	-1.18707	-0.00113135	2.06428	2.38126	0.416 (16)
CP9	0.437254	-0.309421	3.38977	3.43183	0.407 (19)
Total	1.3872	1.02711	4.0004	4.35689	0.348

Table 8. Check points.
X - Easting, Y - Northing, Z - Altitude.

Digital Elevation Model

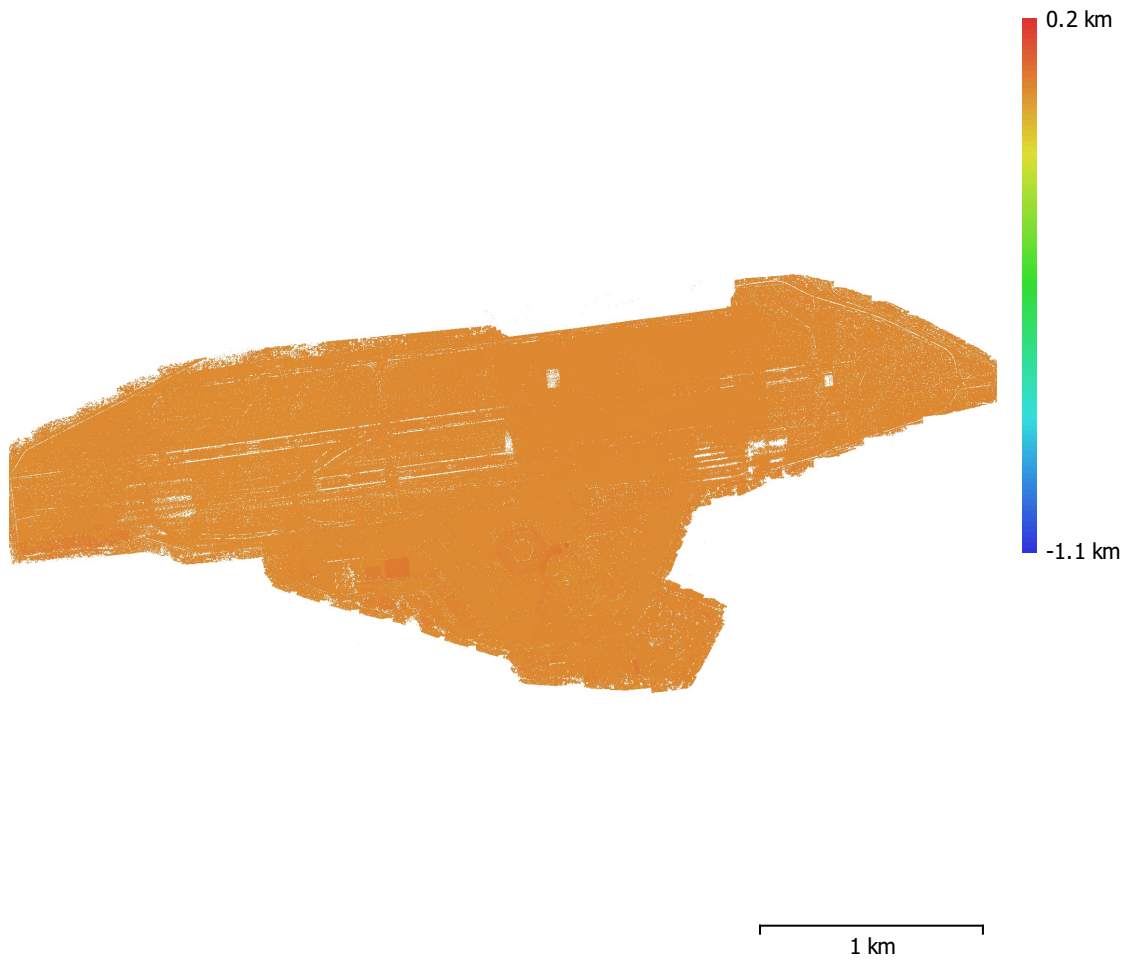


Fig. 6. Reconstructed digital elevation model.

Resolution: unknown
Point density: unknown

Processing Parameters

General

Cameras	10064
Aligned cameras	10054
Markers	32
Coordinate system	ETRS89 / UTM zone 33N + DHHN2016
Rotation angles	Yaw, Pitch, Roll

Tie Points

Points	9,169,276 of 13,361,605
RMS reprojection error	0.181259 (0.680457 pix)
Max reprojection error	4.72334 (62.3585 pix)
Mean key point size	2.84901 pix
Point colors	3 bands, uint8
Key points	No
Average tie point multiplicity	8.36112

Alignment parameters

Accuracy	High
Generic preselection	Yes
Reference preselection	Source
Key point limit	40,000
Key point limit per Mpx	40,000
Tie point limit	10,000
Exclude stationary tie points	No
Guided image matching	Yes
Adaptive camera model fitting	Yes
Matching time	8 hours 35 minutes
Matching memory usage	60.73 GB
Alignment time	2 hours 8 minutes
Alignment memory usage	9.43 GB

Optimization parameters

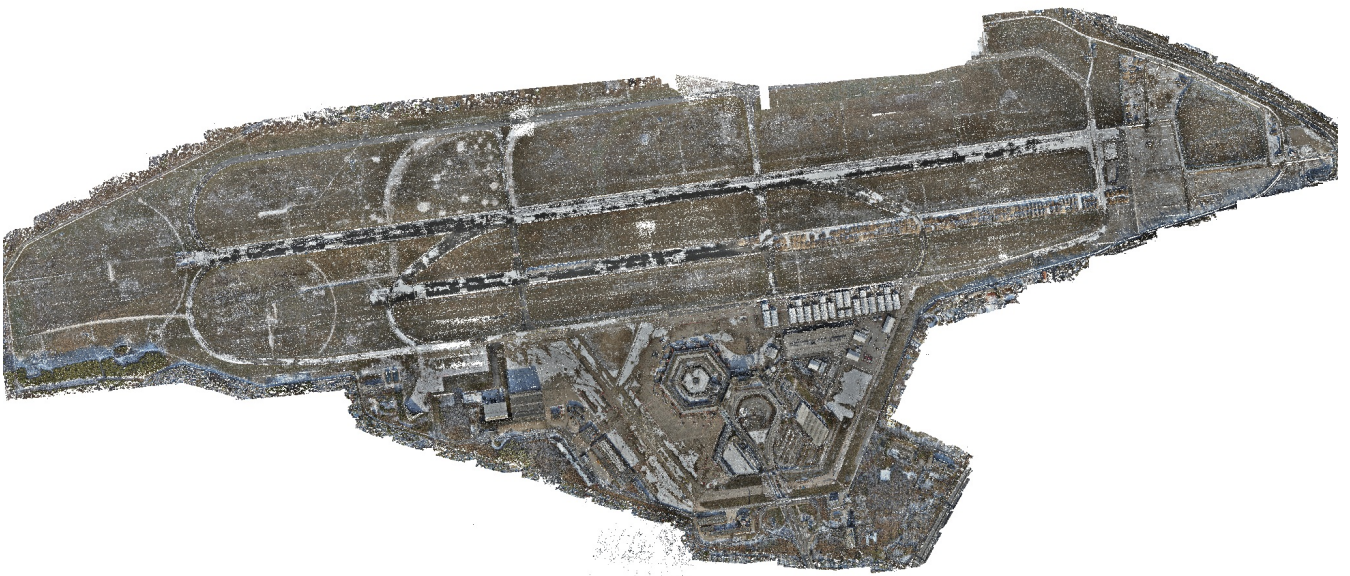
Parameters	f, cx, cy, k1-k4, p1, p2
Fit additional corrections	Yes
Adaptive camera model fitting	No
Optimization time	10 minutes 7 seconds
Date created	2023:03:04 06:22:35
Software version	2.0.0.15597
File size	1.91 GB

System

Software name	Agisoft Metashape Professional
Software version	2.0.0 build 15597
OS	Windows 64 bit
RAM	127.90 GB
CPU	AMD Ryzen 9 5950X 16-Core Processor
GPU(s)	NVIDIA GeForce RTX 3090

Agisoft Metashape

Processing Report
10 March 2023



Survey Data

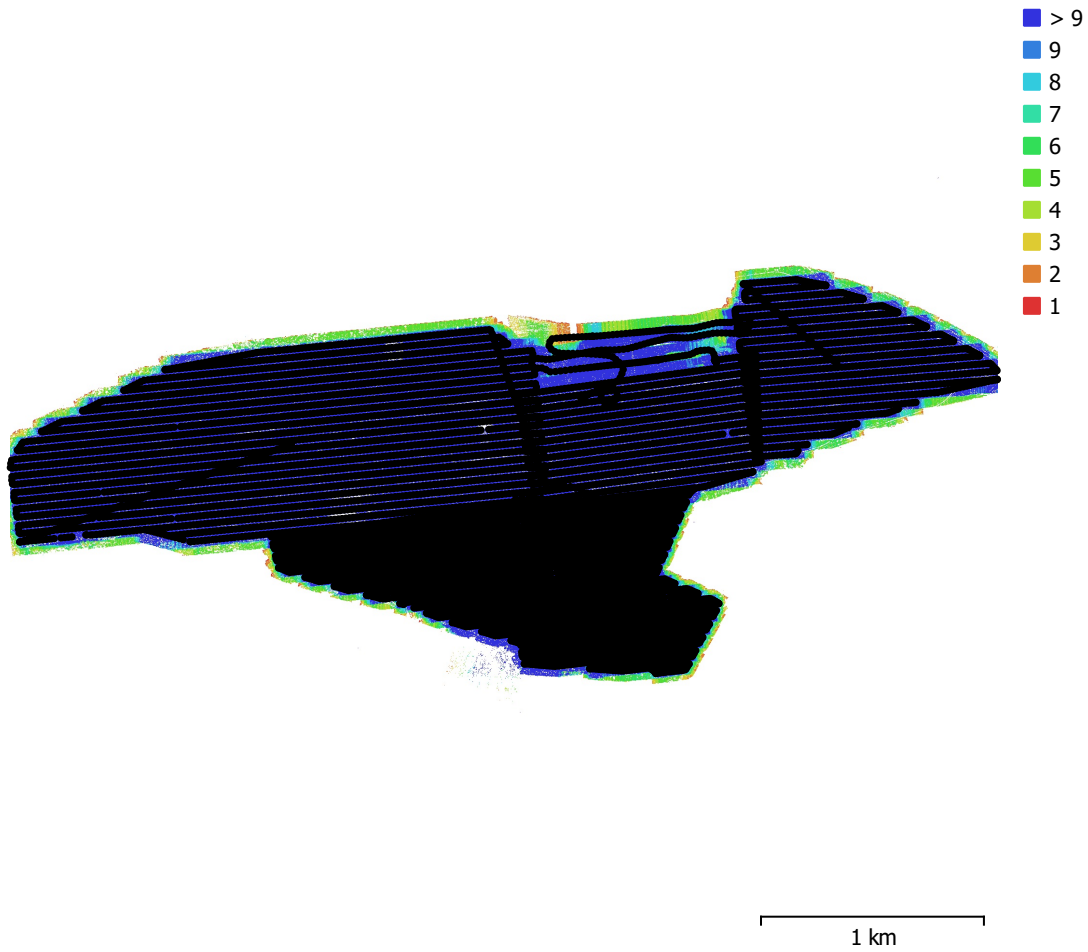


Fig. 1. Camera locations and image overlap.

Number of images:	8,714	Camera stations:	8,711
Flying altitude:	123 m	Tie points:	7,086,014
Ground resolution:	1.49 cm/pix	Projections:	70,436,826
Coverage area:	4.26 km ²	Reprojection error:	0.492 pix

Camera Model	Resolution	Focal Length	Pixel Size	Precalibrated
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No

Table 1. Cameras.

Camera Calibration

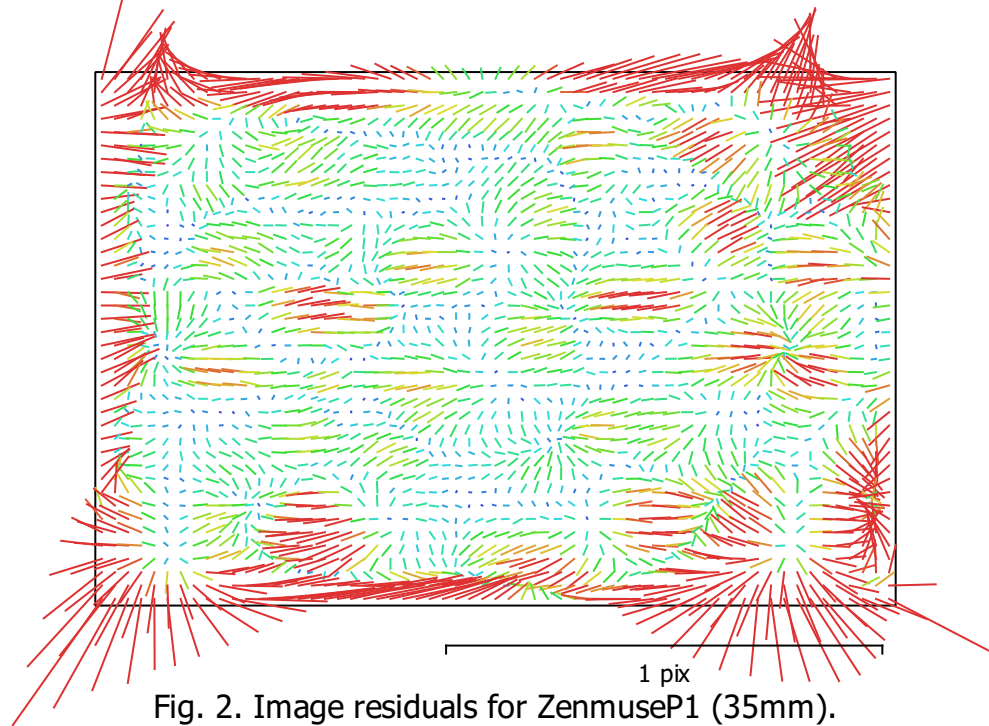


Fig. 2. Image residuals for ZenmuseP1 (35mm).

ZenmuseP1 (35mm)

3162 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8196.41	0.04	1.00	-0.02	-0.03	-0.98	0.97	-0.94	0.91	-0.01	-0.04
Cx	-11.6339	0.016		1.00	0.01	0.02	-0.02	0.01	-0.00	0.98	-0.00
Cy	39.6426	0.017			1.00	0.04	-0.04	0.03	-0.03	0.00	0.99
K1	-0.0479739	8.7e-05				1.00	-0.99	0.98	-0.95	0.01	0.05
K2	0.160498	0.00057					1.00	-0.99	0.98	-0.01	-0.04
K3	-0.526657	0.0016						1.00	-0.99	0.00	0.04
K4	0.462027	0.0017							1.00	0.00	-0.04
P1	-0.00113012	9.2e-07								1.00	-0.01
P2	0.00233636	9.9e-07									1.00

Table 2. Calibration coefficients and correlation matrix.

Camera Calibration

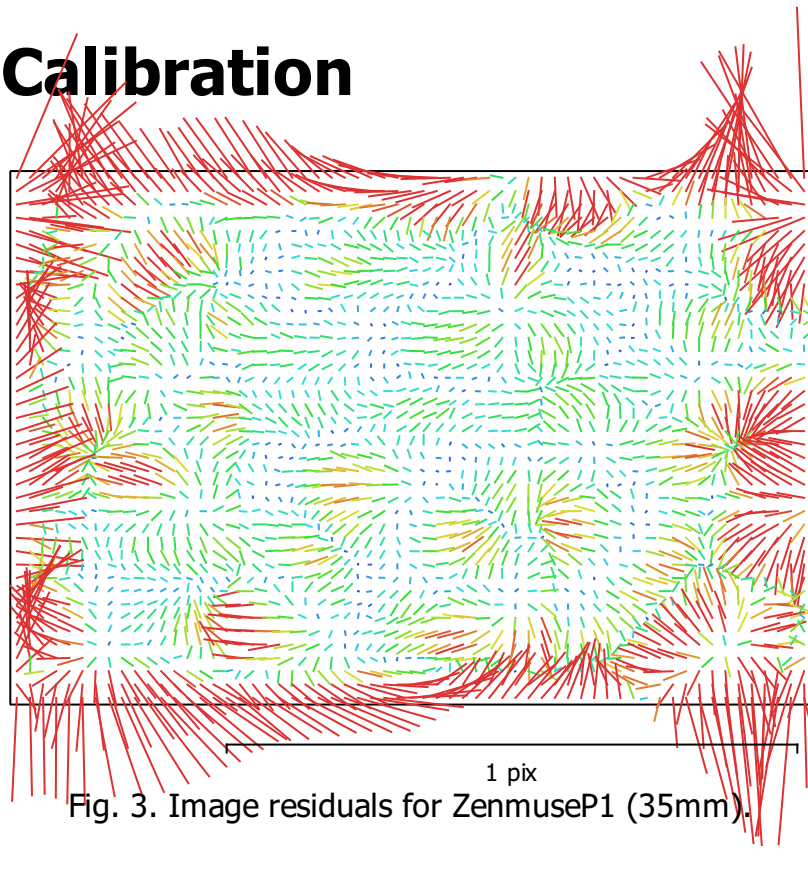


Fig. 3. Image residuals for ZenmuseP1 (35mm).

ZenmuseP1 (35mm)

5552 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8203.24	0.031	1.00	-0.05	-0.01	-0.98	0.96	-0.94	0.91	-0.05	-0.02
Cx	-32.4045	0.013		1.00	0.02	0.06	-0.06	0.06	-0.06	0.98	0.01
Cy	28.7839	0.013			1.00	0.02	-0.02	0.02	-0.01	0.01	0.99
K1	-0.0628952	6.7e-05				1.00	-0.99	0.98	-0.95	0.05	0.03
K2	0.252499	0.00043					1.00	-0.99	0.98	-0.05	-0.02
K3	-0.787995	0.0012						1.00	-0.99	0.05	0.02
K4	0.733275	0.0013							1.00	-0.05	-0.01
P1	-0.00263156	7.3e-07								1.00	0.01
P2	0.0014466	7.5e-07									1.00

Table 3. Calibration coefficients and correlation matrix.

Camera Locations

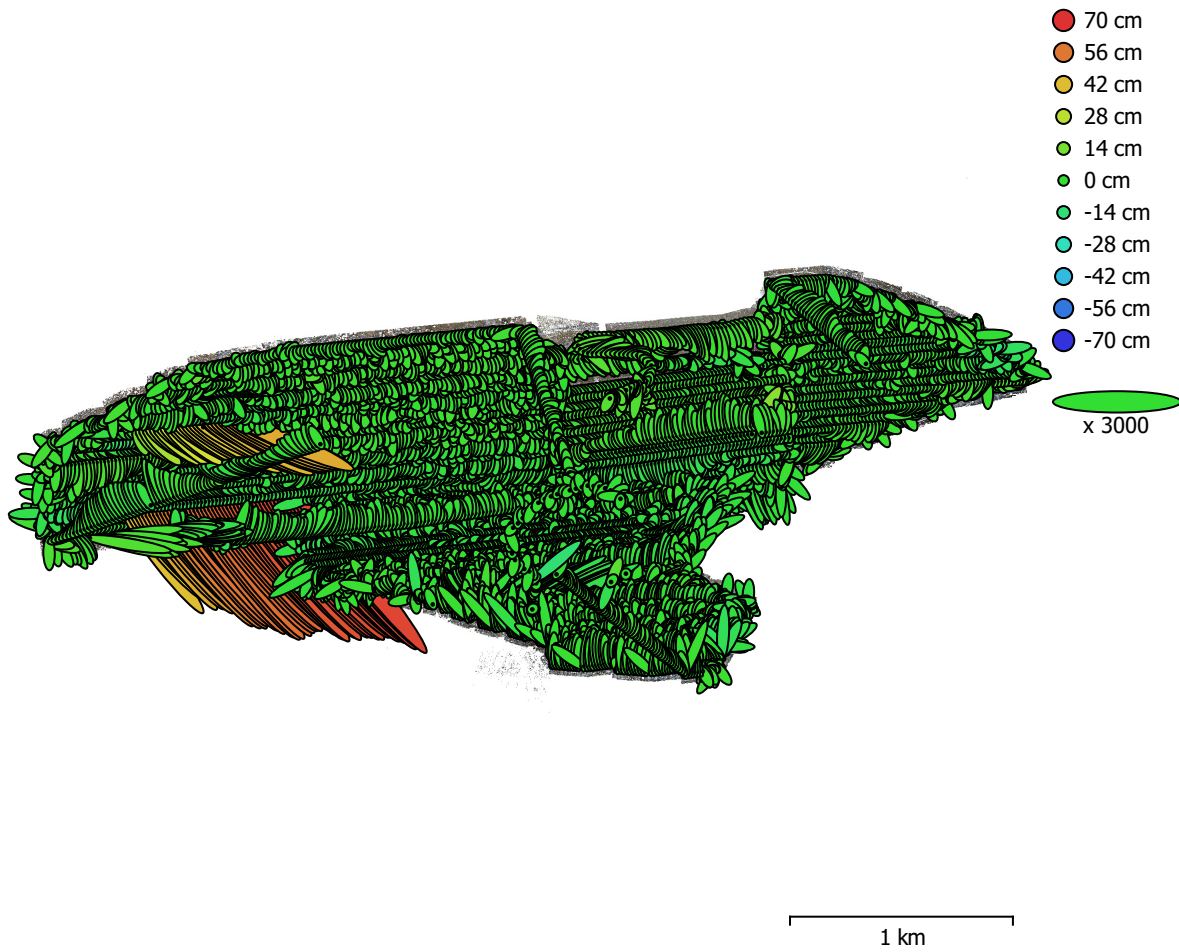


Fig. 4. Camera locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated camera locations are marked with a black dot.

X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total error (cm)
1.82258	2.16953	5.60244	2.83349	6.27821

Table 4. Average camera location error.

X - Easting, Y - Northing, Z - Altitude.

Ground Control Points



Fig. 5. GCP locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape. Estimated GCP locations are marked with a dot or crossing.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
20	1.41154	2.01791	2.43076	2.4626	3.4602

Table 5. Control points RMSE.
X - Easting, Y - Northing, Z - Altitude.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
9	1.46646	3.2883	5.28517	3.60047	6.39503

Table 6. Check points RMSE.
X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
1	0.608353	-1.91876	3.3179	3.88074	1.116 (15)
2	-0.580059	-1.03952	1.74057	2.10871	0.945 (16)
3	0.648917	-0.760602	2.91513	3.08182	1.571 (7)
4	0.99243	-2.04061	2.49061	3.3693	0.945 (15)
5	1.96804	-4.4977	4.0959	6.39366	1.222 (15)
6	0.360143	-2.67578	1.18076	2.94682	1.132 (5)
7	1.86666	-2.76745	-0.350458	3.35649	0.806 (15)
8	-2.21136	-3.49144	0.00589162	4.13283	0.818 (13)
9	0.403164	-1.56073	0.525662	1.6955	0.839 (14)
10	-1.2632	-2.47989	-2.52176	3.75564	1.228 (14)
11					
12	1.39481	-1.05047	-0.0245991	1.7463	2.548 (1)
13	1.25916	-1.58557	0.731922	2.15295	0.817 (15)
14	1.8594	-1.78811	5.9778	6.51067	1.448 (17)
15					
16					
17	-3.16628	-0.841346	2.02104	3.84939	0.955 (15)
18	1.53752	-1.56659	1.33898	2.57119	0.973 (7)
19	1.37534	-0.926455	3.377	3.76218	0.959 (21)
20	0.781474	-0.190832	0.336014	0.871794	0.754 (10)
21	-0.623291	-2.14086	2.935	3.68592	0.929 (20)
22	1.13127	-0.78764	-0.971531	1.68642	0.778 (21)
23	0.228753	-0.187658	1.1316	1.16964	0.828 (17)
Total	1.41154	2.01791	2.43076	3.4602	1.014

Table 7. Control points.
X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP1	-1.66971	0.636433	8.52147	8.7068	0.354 (17)
CP2	-0.0555916	-1.58233	4.12679	4.4201	0.241 (15)
CP3	-0.931565	-2.0081	4.76723	5.25611	0.319 (16)

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP4	0.45907	-2.84923	5.08028	5.84278	0.326 (15)
CP5	-0.895667	-2.8451	1.84697	3.50829	0.244 (21)
CP6	1.39124	-5.06733	2.04127	5.63739	0.572 (16)
CP7	-1.51661	-5.35801	8.45796	10.1265	0.688 (17)
CP8	0.229999	-3.08456	5.43838	6.25647	0.362 (16)
CP9	3.22401	-3.20336	-2.12711	5.018	0.359 (17)
Total	1.46646	3.2883	5.28517	6.39503	0.409

Table 8. Check points.
X - Easting, Y - Northing, Z - Altitude.

Digital Elevation Model

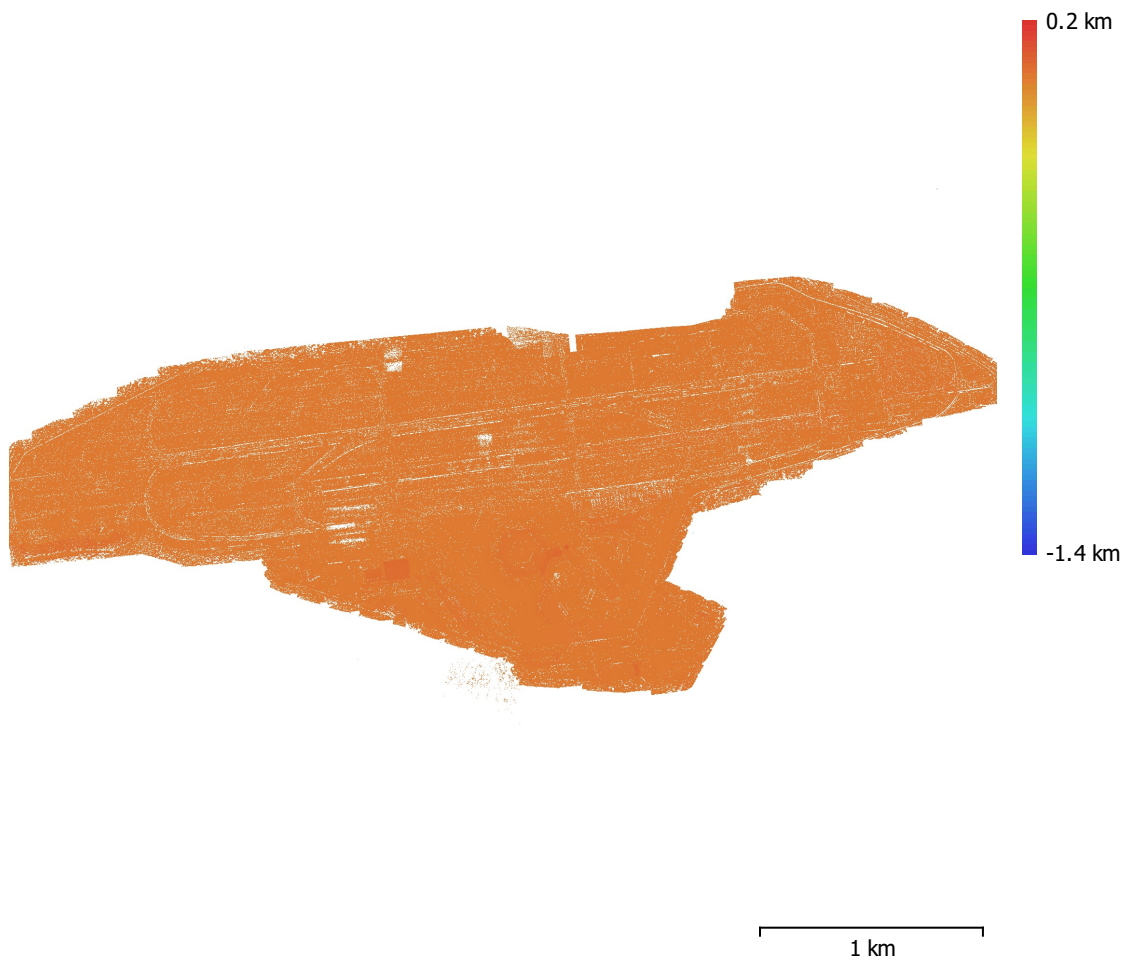


Fig. 6. Reconstructed digital elevation model.

Resolution: unknown
Point density: unknown

Processing Parameters

General

Cameras	8714
Aligned cameras	8711
Markers	32
Coordinate system	ETRS89 / UTM zone 33N + DHHN2016
Rotation angles	Yaw, Pitch, Roll

Tie Points

Points	7,086,014 of 9,810,279
RMS reprojection error	0.156226 (0.492391 pix)
Max reprojection error	1.47172 (56.0507 pix)
Mean key point size	2.56412 pix
Point colors	3 bands, uint8
Key points	No
Average tie point multiplicity	9.75774

Alignment parameters

Accuracy	High
Generic preselection	Yes
Reference preselection	Source
Key point limit	40,000
Key point limit per Mpx	40,000
Tie point limit	10,000
Exclude stationary tie points	No
Guided image matching	Yes
Adaptive camera model fitting	Yes
Matching time	7 hours 24 minutes
Matching memory usage	53.60 GB
Alignment time	2 hours 0 minutes
Alignment memory usage	7.47 GB

Optimization parameters

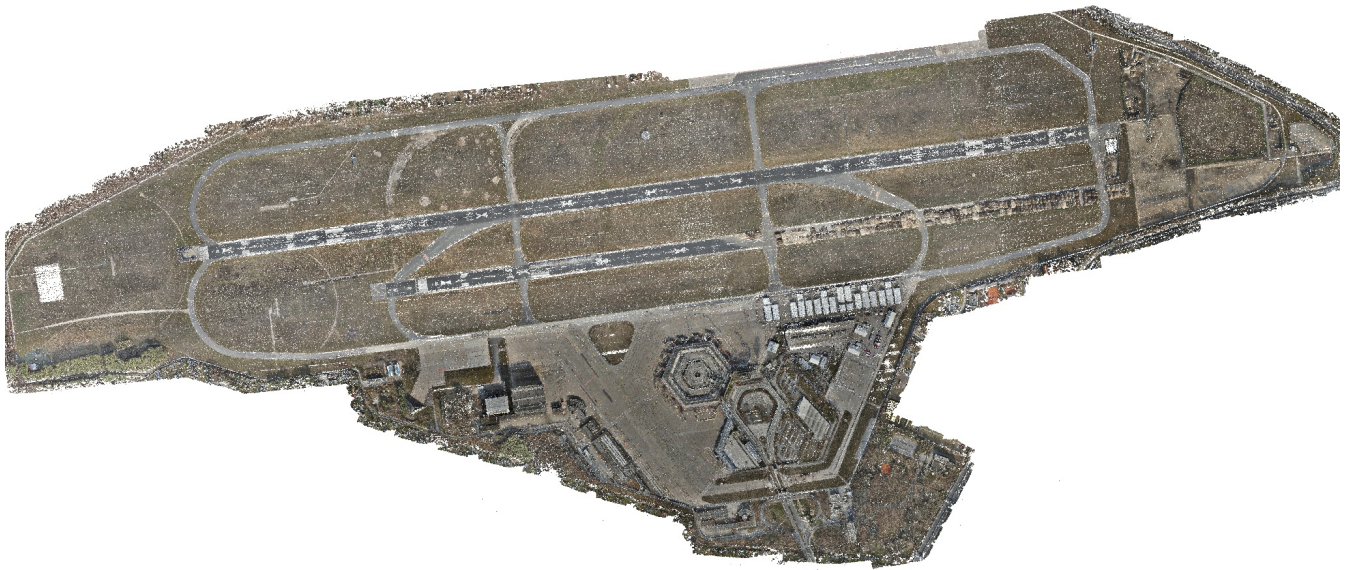
Parameters	f, cx, cy, k1-k4, p1, p2
Fit additional corrections	Yes
Adaptive camera model fitting	No
Optimization time	26 minutes 42 seconds
Date created	2023:03:05 04:54:54
Software version	2.0.0.15597
File size	1.61 GB

System

Software name	Agisoft Metashape Professional
Software version	2.0.0 build 15597
OS	Windows 64 bit
RAM	127.90 GB
CPU	AMD Ryzen 9 5950X 16-Core Processor
GPU(s)	NVIDIA GeForce RTX 3090

Agisoft Metashape

Processing Report
10 March 2023



Survey Data

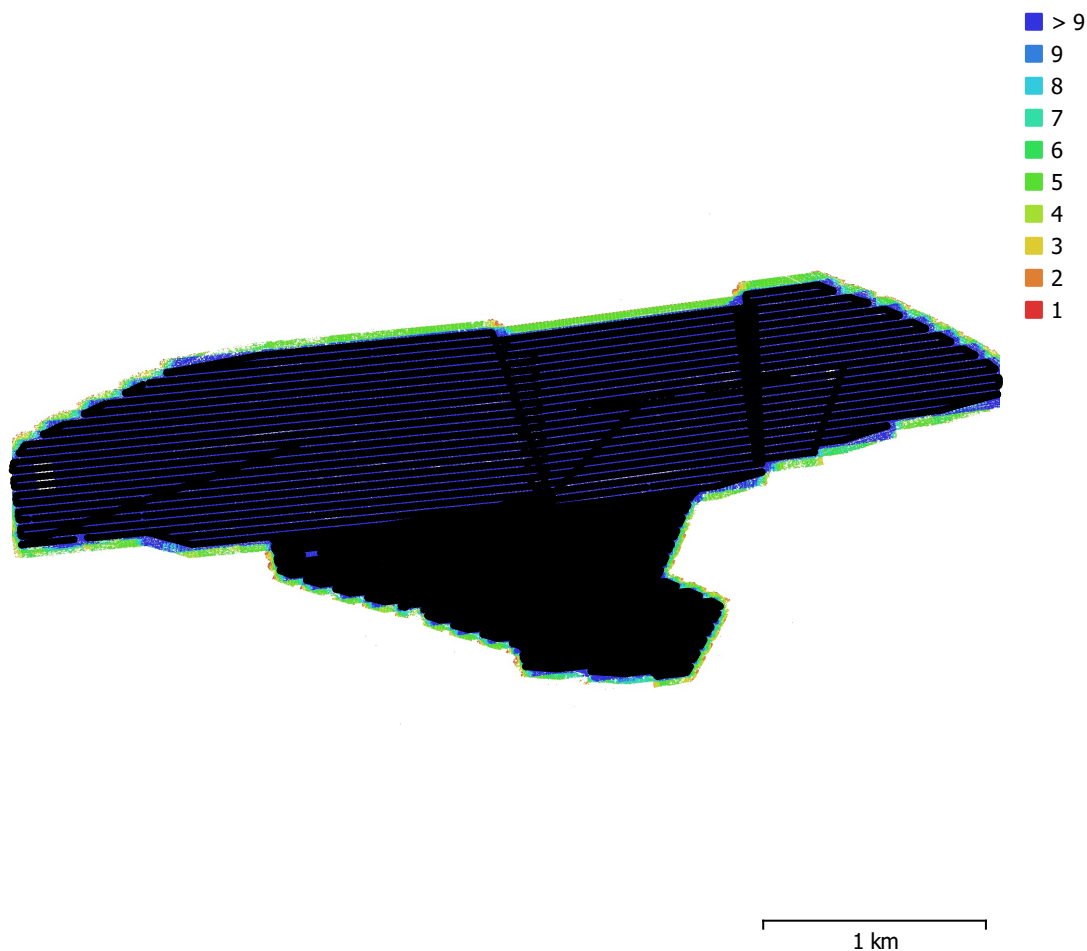


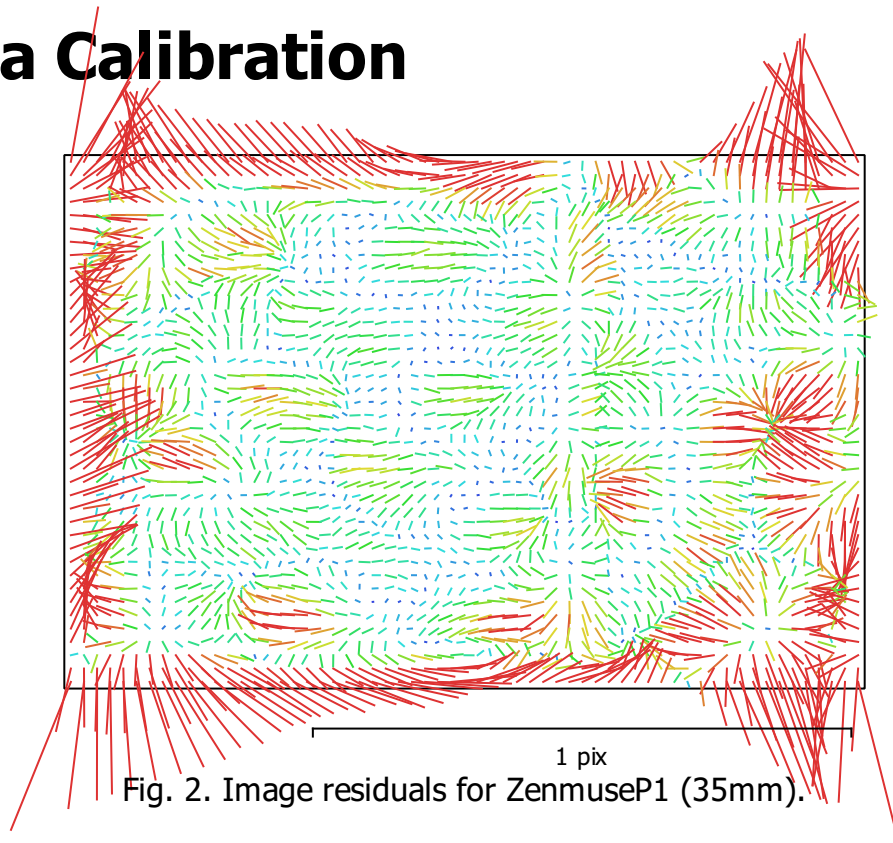
Fig. 1. Camera locations and image overlap.

Number of images:	9,289	Camera stations:	9,281
Flying altitude:	122 m	Tie points:	7,564,942
Ground resolution:	1.49 cm/pix	Projections:	71,748,070
Coverage area:	4.37 km ²	Reprojection error:	0.607 pix

Camera Model	Resolution	Focal Length	Pixel Size	Precalibrated
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No

Table 1. Cameras.

Camera Calibration



ZenmuseP1 (35mm)

4692 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8199.93	0.036	1.00	-0.08	-0.01	-0.98	0.97	-0.94	0.92	-0.07	-0.02
Cx	-29.4786	0.015		1.00	0.02	0.08	-0.09	0.09	-0.09	0.98	0.02
Cy	25.6392	0.015			1.00	0.02	-0.02	0.01	-0.01	0.02	0.99
K1	-0.0496773	7.7e-05				1.00	-1.00	0.98	-0.96	0.07	0.02
K2	0.152494	0.0005					1.00	-0.99	0.98	-0.08	-0.02
K3	-0.489473	0.0014						1.00	-1.00	0.08	0.02
K4	0.415866	0.0015							1.00	-0.08	-0.01
P1	-0.00243026	8.3e-07								1.00	0.02
P2	0.00142007	8.4e-07									1.00

Table 2. Calibration coefficients and correlation matrix.

Camera Calibration

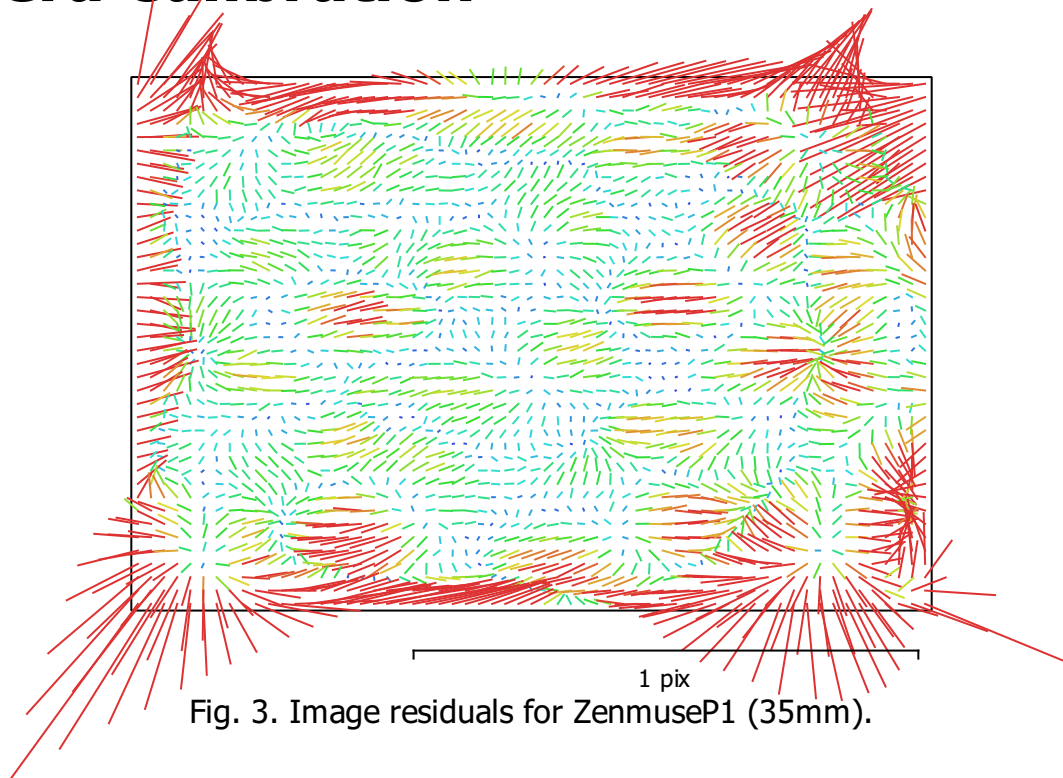


Fig. 3. Image residuals for ZenmuseP1 (35mm).

ZenmuseP1 (35mm)

4597 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8200.5	0.044	1.00	-0.01	-0.02	-0.99	0.97	-0.95	0.91	-0.00	-0.03
Cx	-11.0238	0.019		1.00	0.01	0.01	-0.00	0.00	0.00	0.98	0.00
Cy	36.0288	0.02			1.00	0.03	-0.03	0.03	-0.02	0.00	0.99
K1	-0.052512	9.6e-05				1.00	-0.99	0.98	-0.95	-0.00	0.04
K2	0.187752	0.00063					1.00	-0.99	0.98	0.00	-0.04
K3	-0.607114	0.0018						1.00	-0.99	-0.00	0.03
K4	0.548679	0.0019							1.00	0.01	-0.03
P1	-0.000924181	1.1e-06								1.00	-0.01
P2	0.00230955	1.1e-06									1.00

Table 3. Calibration coefficients and correlation matrix.

Camera Locations

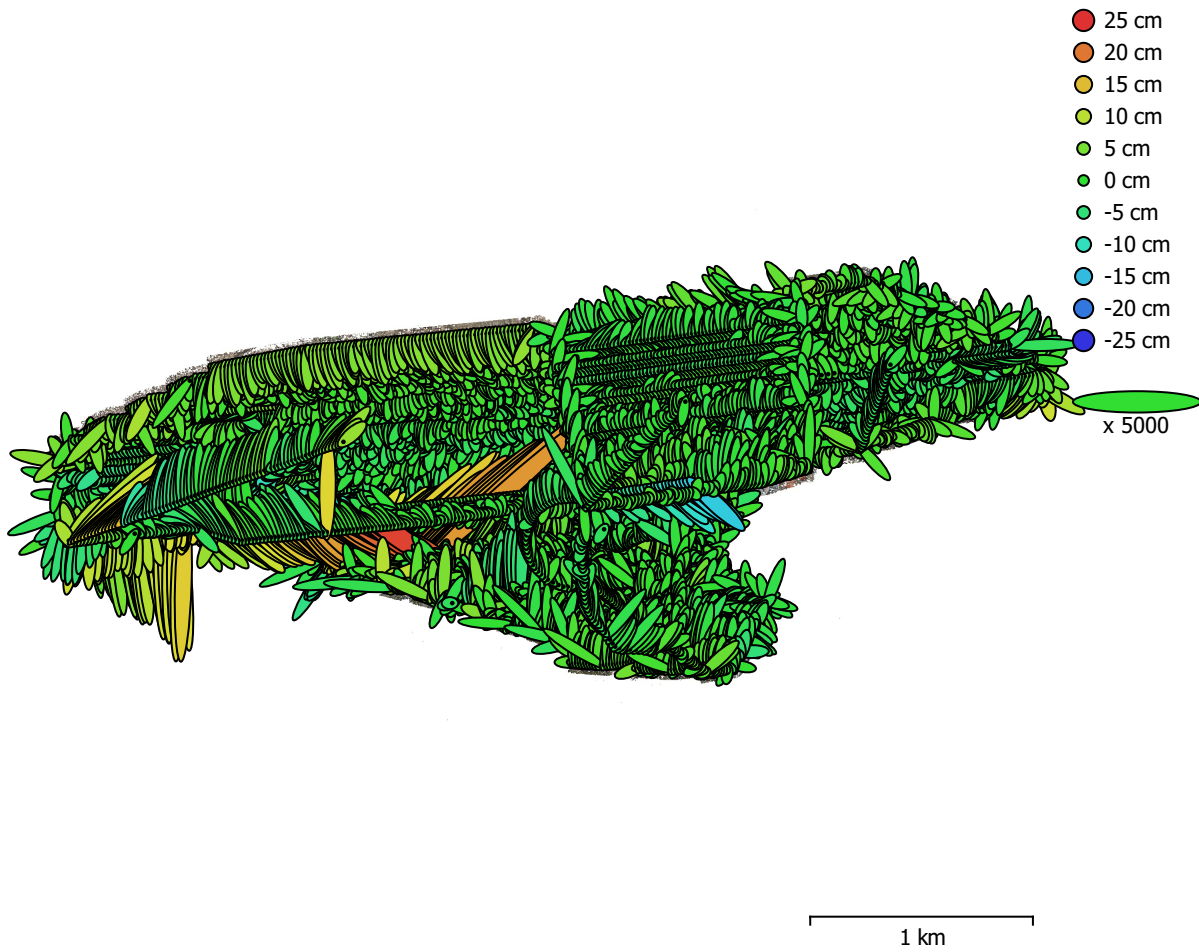


Fig. 4. Camera locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated camera locations are marked with a black dot.

X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total error (cm)
1.18893	1.61467	3.18388	2.00517	3.76268

Table 4. Average camera location error.

X - Easting, Y - Northing, Z - Altitude.

Ground Control Points



Fig. 5. GCP locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated GCP locations are marked with a dot or crossing.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
23	1.58254	1.7769	1.76754	2.37945	2.96412

Table 5. Control points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
9	1.4909	2.06365	5.04389	2.54586	5.64997

Table 6. Check points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
1	0.841832	-1.38001	2.39582	2.89017	1.036 (14)
2	-0.027034	-0.537401	0.621412	0.821999	0.663 (15)
3	1.8096	0.376679	2.80957	3.36307	0.983 (20)
4	0.636725	-1.32151	1.68098	2.23104	0.890 (14)
5	1.42958	-2.96086	2.42349	4.08457	1.004 (15)
6	2.50471	-2.2724	1.0001	3.5267	0.885 (10)
7	2.41076	-1.37354	-1.04787	2.96587	0.899 (15)
8	-0.802176	-2.92365	-2.01269	3.63897	1.290 (13)
9	-0.871428	0.834789	0.370754	1.26243	0.883 (20)
10	-2.01661	-1.17981	-1.97402	3.05866	1.033 (14)
11	-2.50322	2.28947	2.48771	4.20672	1.288 (12)
12	2.46155	-1.3328	0.559004	2.85448	0.743 (15)
13	1.44562	-1.02428	0.0474331	1.77235	0.820 (15)
14	0.693874	-0.315017	3.012	3.1069	1.126 (15)
15	-1.29504	-1.80961	0.585795	2.30109	0.748 (10)
16	-2.35494	-4.5018	-1.45314	5.28427	0.790 (14)
17	-1.87108	0.016359	2.0515	2.77667	0.900 (15)
18	1.91625	-1.25298	2.21686	3.18691	1.148 (11)
19	1.75792	-0.730385	0.973653	2.13817	0.870 (20)
20	0.325395	1.52954	0.666998	1.70008	0.778 (15)
21	-0.392865	-1.68115	2.66194	3.17278	1.034 (17)
22	0.946631	-0.179539	-1.82249	2.06151	0.839 (20)
23	0.272944	1.75359	0.0269646	1.77491	0.815 (16)
Total	1.58254	1.7769	1.76754	2.96412	0.940

Table 7. Control points.
X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP1	-2.521	2.64866	9.01417	9.7276	0.258 (18)
CP2	0.603066	-1.60103	1.39466	2.20727	0.282 (15)
CP3	-1.89402	-0.144682	5.19158	5.52817	0.320 (17)

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP4	0.0586923	-2.30283	5.75523	6.19912	0.272 (21)
CP5	0.02154	-0.364517	4.30994	4.32538	0.273 (21)
CP6	-0.525279	-2.86066	3.91841	4.87987	0.578 (15)
CP7	-1.18084	-2.34404	6.83557	7.32215	0.359 (20)
CP8	0.269636	-2.4139	2.24433	3.30706	0.468 (18)
CP9	2.81986	-1.94617	-0.0107497	3.42626	0.327 (18)
Total	1.4909	2.06365	5.04389	5.64997	0.356

Table 8. Check points.
X - Easting, Y - Northing, Z - Altitude.

Digital Elevation Model

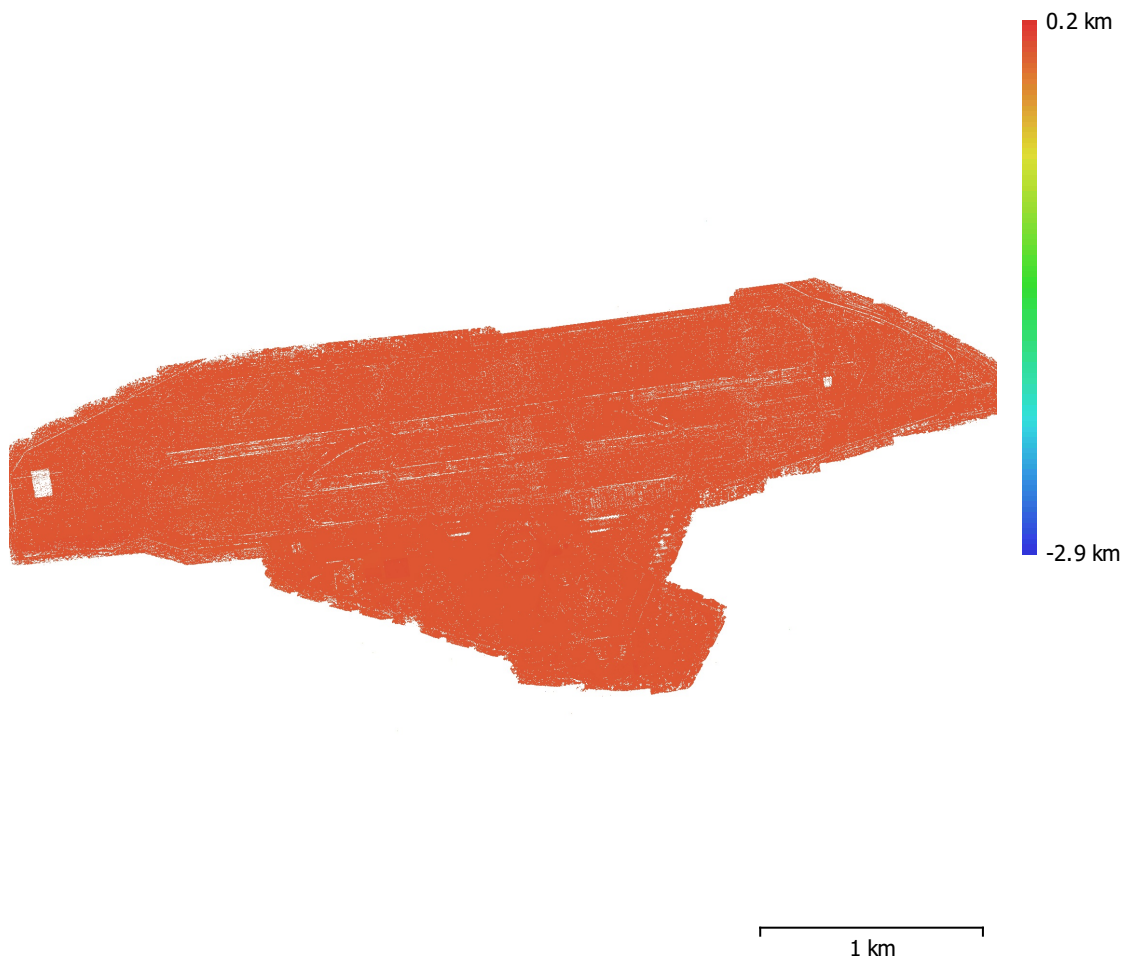


Fig. 6. Reconstructed digital elevation model.

Resolution: unknown
Point density: unknown

Processing Parameters

General

Cameras	9289
Aligned cameras	9281
Markers	32
Coordinate system	ETRS89 / UTM zone 33N + DHHN2016
Rotation angles	Yaw, Pitch, Roll

Tie Points

Points	7,564,942 of 10,979,718
RMS reprojection error	0.17253 (0.607133 pix)
Max reprojection error	1.56127 (62.4591 pix)
Mean key point size	2.80527 pix
Point colors	3 bands, uint8
Key points	No
Average tie point multiplicity	9.27674

Alignment parameters

Accuracy	High
Generic preselection	Yes
Reference preselection	Source
Key point limit	40,000
Key point limit per Mpx	40,000
Tie point limit	10,000
Exclude stationary tie points	No
Guided image matching	Yes
Adaptive camera model fitting	Yes
Matching time	8 hours 0 minutes
Matching memory usage	60.41 GB
Alignment time	2 hours 18 minutes
Alignment memory usage	8.24 GB

Optimization parameters

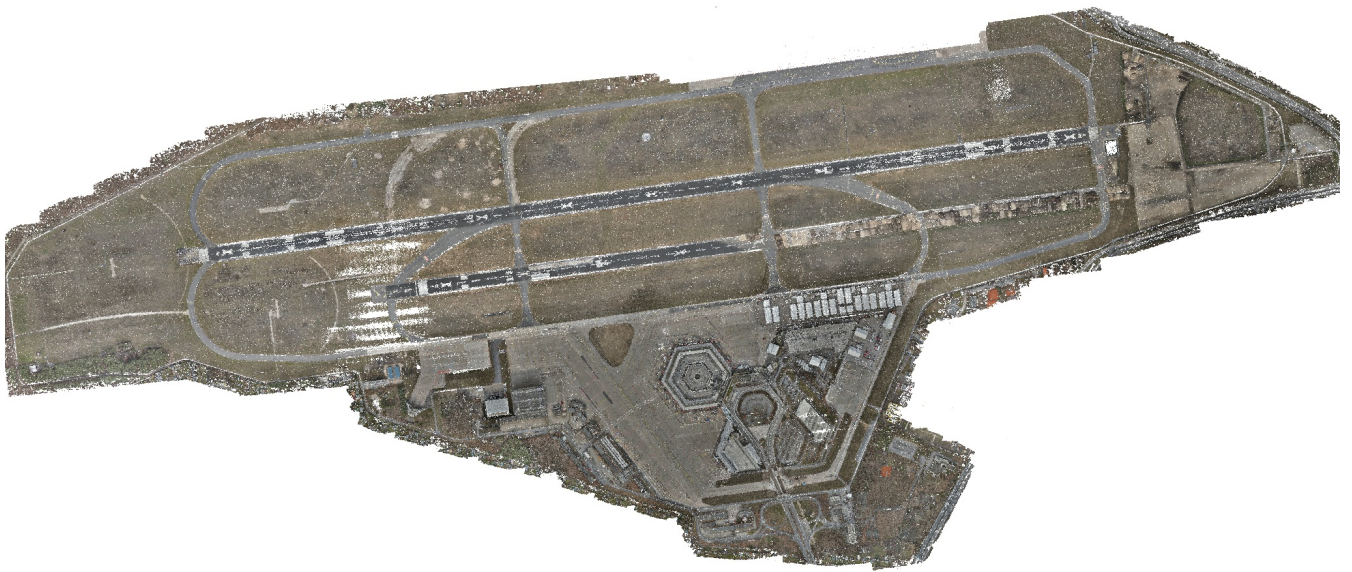
Parameters	f, cx, cy, k1-k4, p1, p2
Fit additional corrections	Yes
Adaptive camera model fitting	No
Optimization time	27 minutes 10 seconds
Date created	2023:03:05 19:43:03
Software version	2.0.0.15597
File size	1.72 GB

System

Software name	Agisoft Metashape Professional
Software version	2.0.0 build 15597
OS	Windows 64 bit
RAM	127.90 GB
CPU	AMD Ryzen 9 5950X 16-Core Processor
GPU(s)	NVIDIA GeForce RTX 3090

Agisoft Metashape

Processing Report
10 March 2023



Survey Data

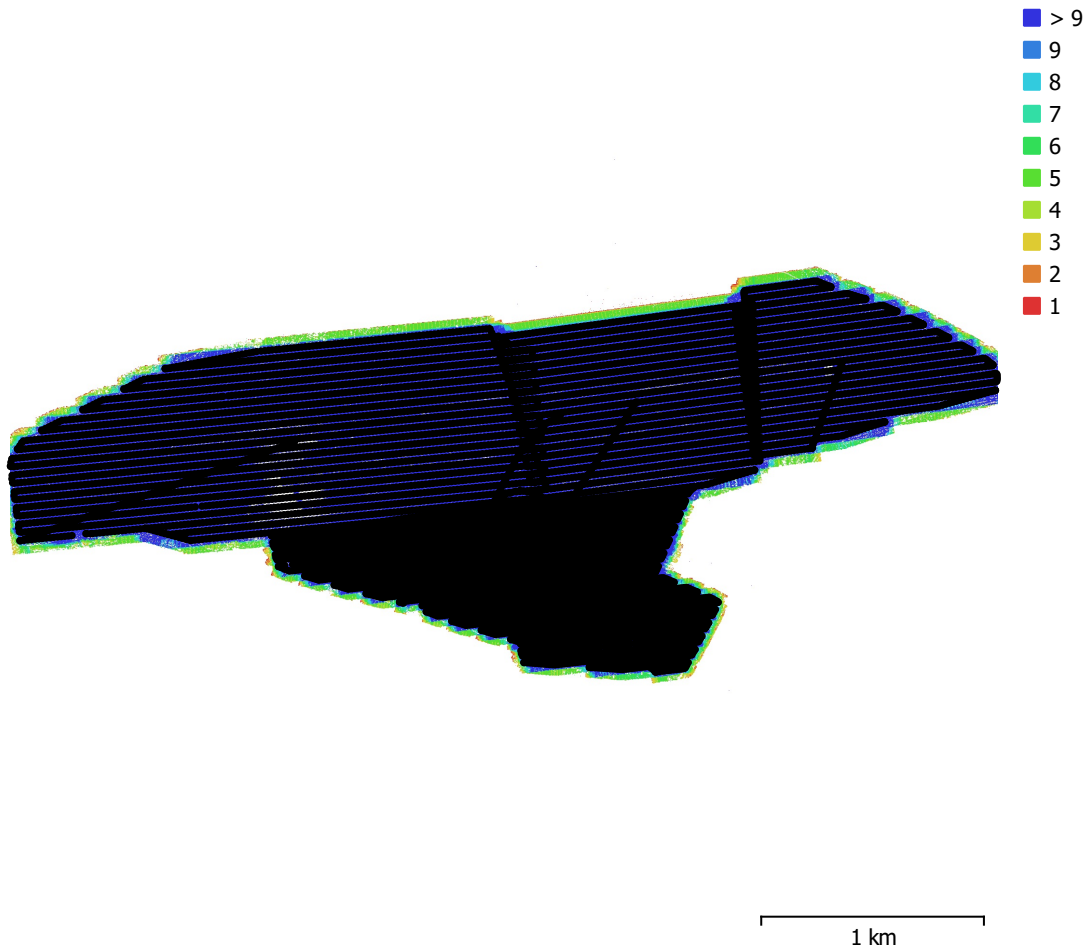


Fig. 1. Camera locations and image overlap.

Number of images:	9,266	Camera stations:	9,266
Flying altitude:	122 m	Tie points:	7,962,571
Ground resolution:	1.49 cm/pix	Projections:	73,992,668
Coverage area:	4.41 km ²	Reprojection error:	0.648 pix

Camera Model	Resolution	Focal Length	Pixel Size	Precalibrated
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No
ZenmuseP1 (35mm)	8192 x 5460	35 mm	4.39 x 4.39 μ m	No

Table 1. Cameras.

Camera Calibration

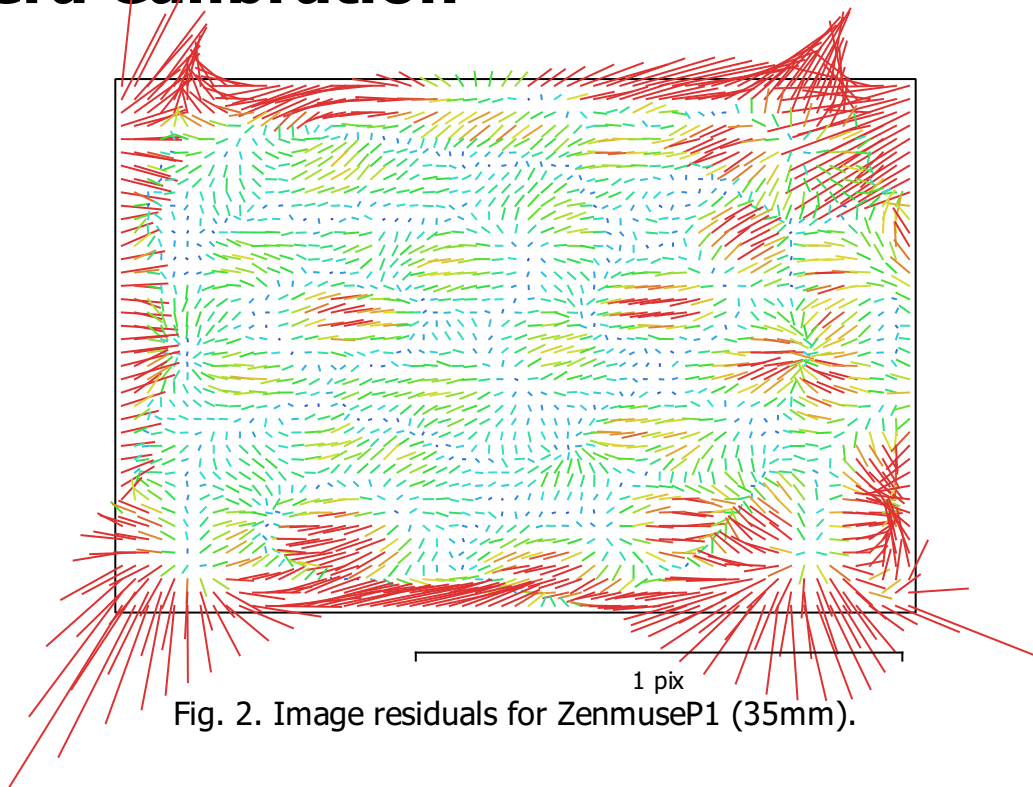


Fig. 2. Image residuals for ZenmuseP1 (35mm).

ZenmuseP1 (35mm)

4583 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8201.57	0.044	1.00	-0.00	-0.03	-0.99	0.97	-0.95	0.92	-0.00	-0.03
Cx	-10.8816	0.019		1.00	0.01	0.00	-0.00	0.00	0.00	0.98	-0.00
Cy	35.7131	0.02			1.00	0.04	-0.04	0.03	-0.03	-0.00	0.99
K1	-0.0540647	9.7e-05				1.00	-0.99	0.98	-0.95	-0.00	0.04
K2	0.200995	0.00064					1.00	-0.99	0.98	0.00	-0.04
K3	-0.65551	0.0018						1.00	-0.99	-0.00	0.04
K4	0.606973	0.0019							1.00	0.01	-0.03
P1	-0.000875234	1.1e-06								1.00	-0.01
P2	0.00228989	1.1e-06									1.00

Table 2. Calibration coefficients and correlation matrix.

Camera Calibration

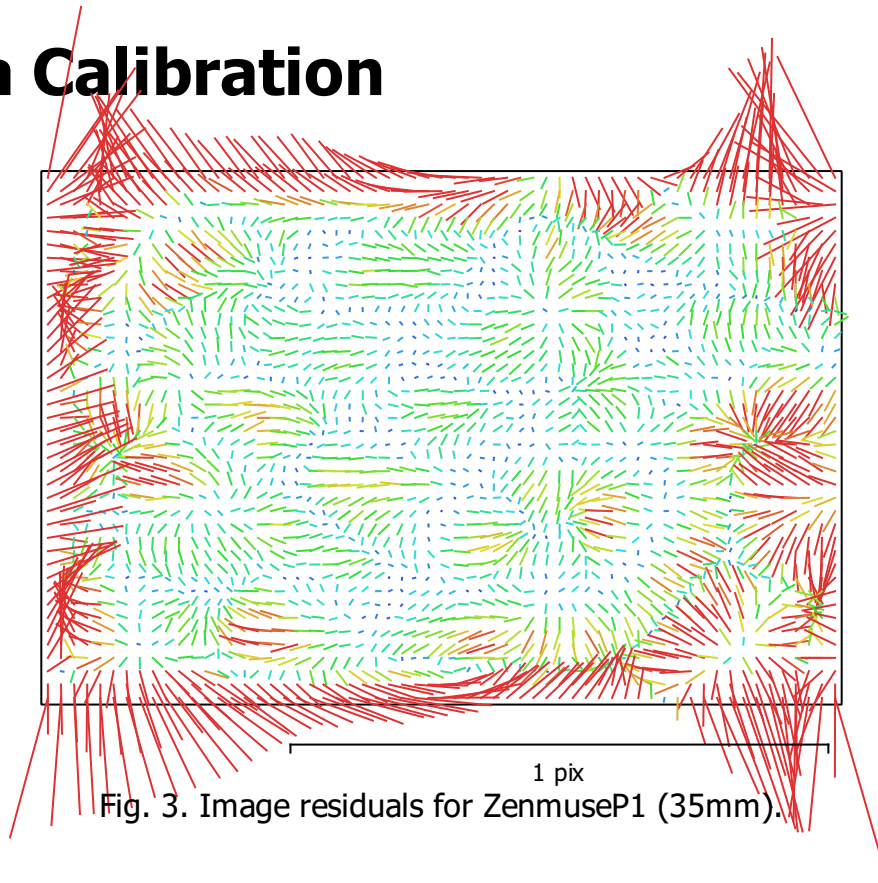


Fig. 3. Image residuals for ZenmuseP1 (35mm).

ZenmuseP1 (35mm)

4683 images, additional corrections

Type	Resolution	Focal Length	Pixel Size
Frame	8192 x 5460	35 mm	4.39 x 4.39 μm

	Value	Error	F	Cx	Cy	K1	K2	K3	K4	P1	P2
F	8203.12	0.038	1.00	-0.07	-0.01	-0.98	0.97	-0.95	0.91	-0.06	-0.02
Cx	-29.9126	0.016		1.00	0.03	0.07	-0.07	0.07	-0.07	0.98	0.02
Cy	25.1365	0.016			1.00	0.02	-0.02	0.01	-0.01	0.02	0.99
K1	-0.0578038	8.2e-05				1.00	-0.99	0.98	-0.96	0.06	0.02
K2	0.208572	0.00054					1.00	-0.99	0.98	-0.07	-0.02
K3	-0.660485	0.0015						1.00	-0.99	0.07	0.02
K4	0.5986	0.0016							1.00	-0.07	-0.01
P1	-0.00240646	8.7e-07								1.00	0.01
P2	0.00138648	9e-07									1.00

Table 3. Calibration coefficients and correlation matrix.

Camera Locations

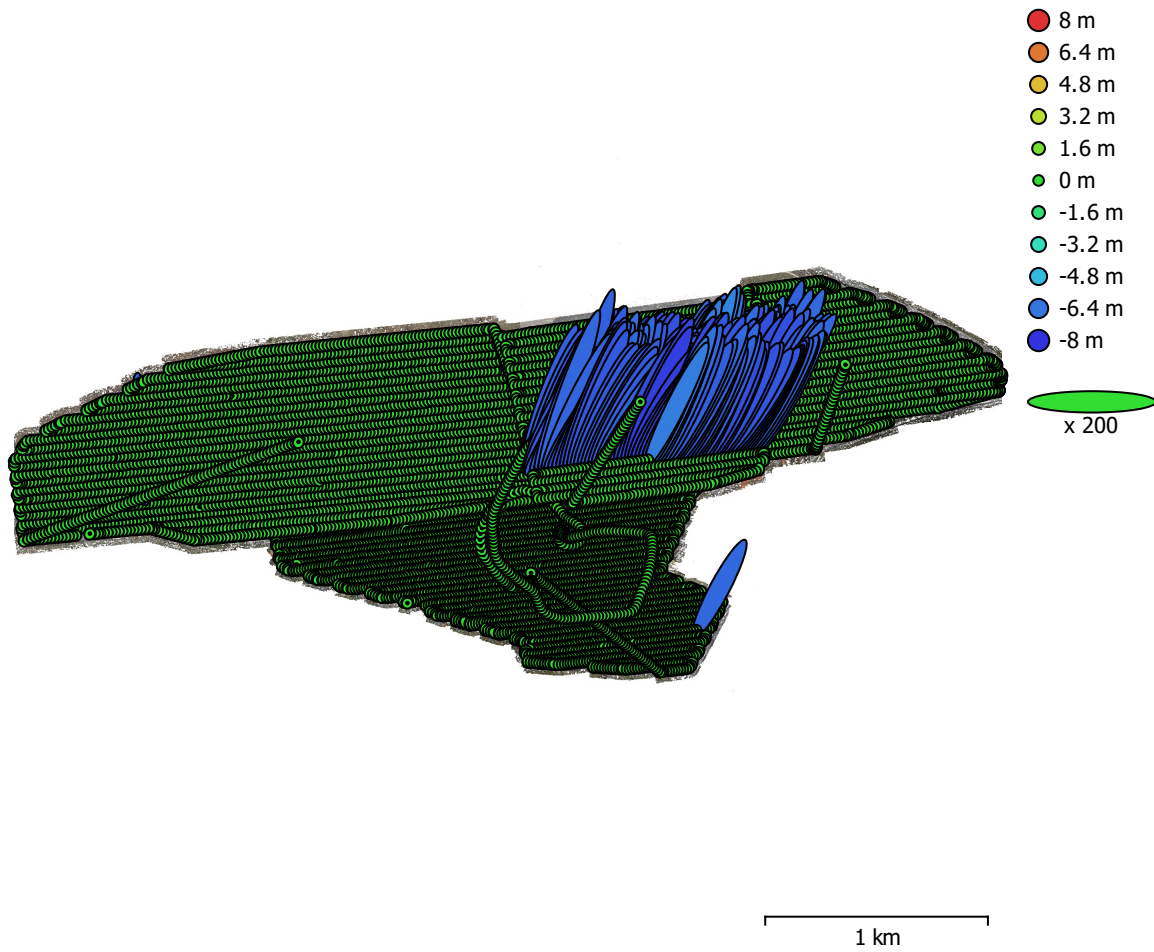


Fig. 4. Camera locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated camera locations are marked with a black dot.

X error (m)	Y error (m)	Z error (m)	XY error (m)	Total error (m)
0.223631	0.437234	1.25536	0.491105	1.34801

Table 4. Average camera location error.

X - Easting, Y - Northing, Z - Altitude.

Ground Control Points

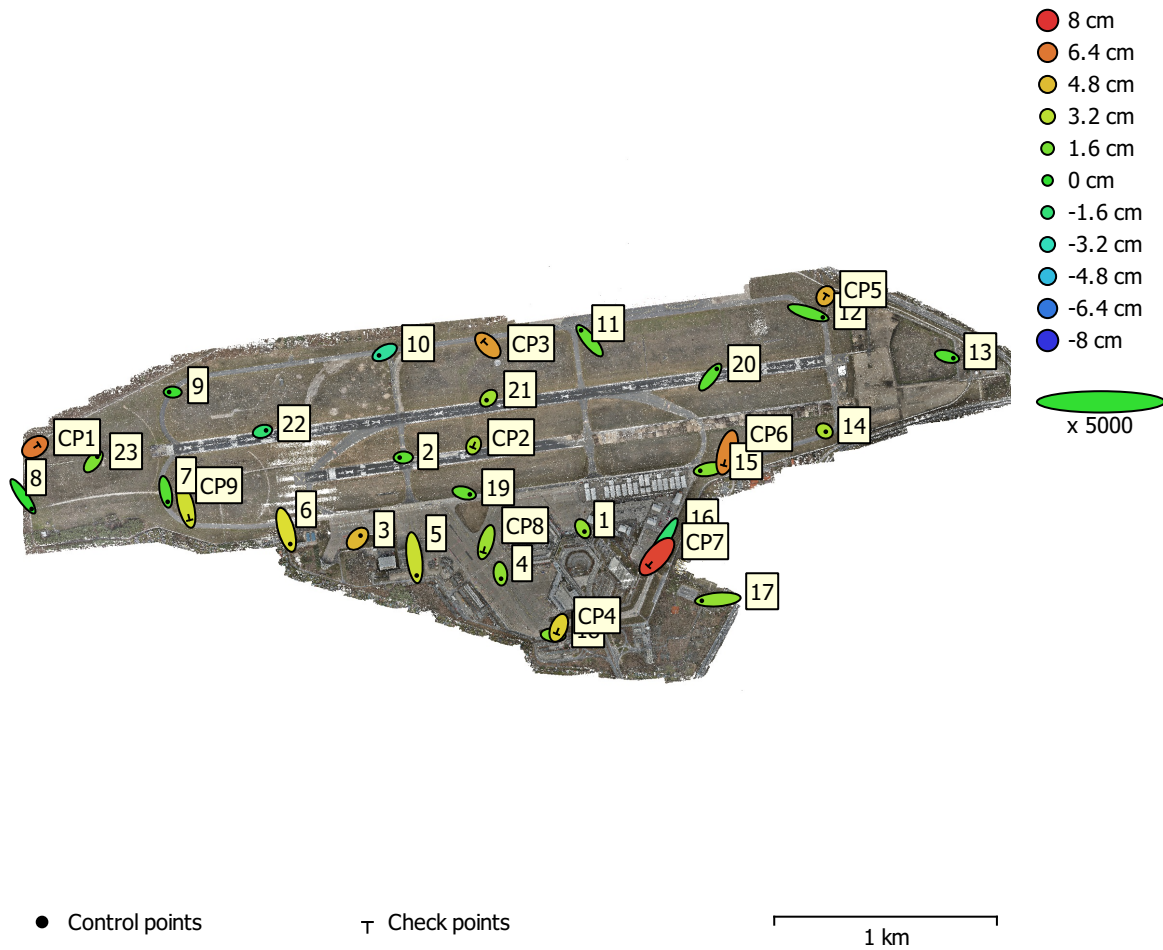


Fig. 5. GCP locations and error estimates.

Z error is represented by ellipse color. X,Y errors are represented by ellipse shape.

Estimated GCP locations are marked with a dot or crossing.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
23	1.26584	1.46671	2.18639	1.93742	2.92128

Table 5. Control points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Count	X error (cm)	Y error (cm)	Z error (cm)	XY error (cm)	Total (cm)
9	0.749603	1.53406	5.13035	1.70741	5.40701

Table 6. Check points RMSE.

X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
1	0.299319	-0.52773	2.27306	2.35264	0.960 (14)
2	-0.652127	-0.0143657	1.05232	1.23809	0.814 (15)
3	0.534095	0.563228	4.93773	4.99837	0.988 (22)
4	0.113062	-0.909548	2.16601	2.35195	0.920 (15)
5	0.413345	-3.20341	3.52139	4.77838	1.291 (14)
6	0.738501	-2.55063	3.98406	4.78788	0.913 (23)
7	0.349292	-1.78974	0.840576	2.00792	0.723 (15)
8	1.5621	-2.25873	-0.0201286	2.74635	0.756 (14)
9	-0.64008	0.013331	0.275476	0.69697	0.848 (15)
10	-1.01066	-0.537801	-2.43673	2.69227	1.107 (14)
11	-1.52726	1.87049	1.26279	2.72505	1.034 (10)
12	2.61762	-0.827639	0.984258	2.91645	0.841 (16)
13	1.13872	-0.305611	0.781329	1.41441	0.714 (15)
14	0.233056	-0.142118	2.81014	2.82336	1.078 (15)
15	-1.60348	-0.284577	1.94164	2.53418	1.019 (15)
16	-2.32364	-3.88099	-1.46167	4.75372	0.841 (15)
17	-2.87518	-0.226985	2.2273	3.64404	0.941 (16)
18	1.05244	-0.0864752	1.8873	2.16264	0.744 (11)
19	1.01672	-0.260831	1.27995	1.6553	0.860 (21)
20	1.16852	1.48092	0.984788	2.12799	0.834 (15)
21	-0.358647	-0.352584	2.54364	2.59288	1.013 (17)
22	0.632792	0.175419	-1.44444	1.5867	0.714 (21)
23	0.723306	0.881952	1.71521	2.05984	0.957 (16)
Total	1.26584	1.46671	2.18639	2.92128	0.916

Table 7. Control points.
X - Easting, Y - Northing, Z - Altitude.

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP1	0.695569	0.44521	6.35203	6.40549	0.284 (18)
CP2	-0.162321	-0.377184	2.68454	2.71576	0.255 (15)
CP3	-0.840571	0.835878	5.54572	5.671	0.249 (15)

Label	X error (cm)	Y error (cm)	Z error (cm)	Total (cm)	Image (pix)
CP4	-0.426181	-1.0665	4.32953	4.47927	0.294 (19)
CP5	0.166731	0.279174	5.12554	5.13585	0.258 (19)
CP6	-0.638548	-2.3425	6.03188	6.5022	0.369 (15)
CP7	-1.56206	-1.73018	7.58886	7.93878	0.346 (18)
CP8	-0.561542	-1.85038	2.45241	3.12307	0.346 (23)
CP9	0.684026	-2.64968	3.68378	4.589	0.224 (17)
Total	0.749603	1.53406	5.13035	5.40701	0.297

Table 8. Check points.
X - Easting, Y - Northing, Z - Altitude.

Digital Elevation Model

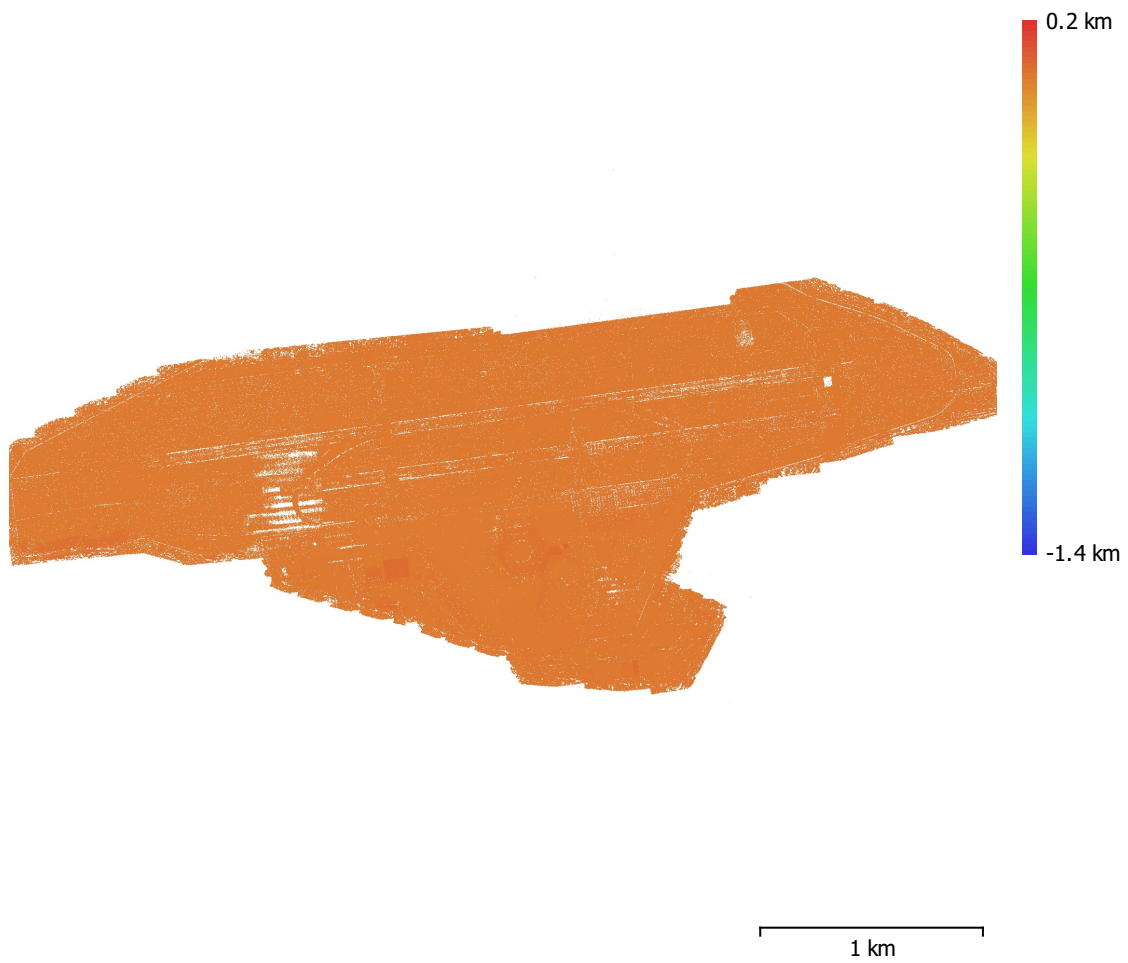


Fig. 6. Reconstructed digital elevation model.

Resolution: unknown
Point density: unknown

Processing Parameters

General

Cameras	9266
Aligned cameras	9266
Markers	32
Coordinate system	ETRS89 / UTM zone 33N + DHHN2016
Rotation angles	Yaw, Pitch, Roll

Tie Points

Points	7,962,571 of 11,098,525
RMS reprojection error	0.173465 (0.647643 pix)
Max reprojection error	2.13047 (59.467 pix)
Mean key point size	2.98766 pix
Point colors	3 bands, uint8
Key points	No
Average tie point multiplicity	9.17345

Alignment parameters

Accuracy	High
Generic preselection	Yes
Reference preselection	Source
Key point limit	40,000
Key point limit per Mpx	40,000
Tie point limit	10,000
Exclude stationary tie points	No
Guided image matching	Yes
Adaptive camera model fitting	Yes
Matching time	7 hours 58 minutes
Matching memory usage	45.59 GB
Alignment time	2 hours 37 minutes
Alignment memory usage	8.01 GB

Optimization parameters

Parameters	f, cx, cy, k1-k4, p1, p2
Fit additional corrections	Yes
Adaptive camera model fitting	No
Optimization time	23 minutes 42 seconds
Date created	2023:03:06 06:15:23
Software version	2.0.0.15597
File size	1.73 GB

System

Software name	Agisoft Metashape Professional
Software version	2.0.0 build 15597
OS	Windows 64 bit
RAM	127.90 GB
CPU	AMD Ryzen 9 5950X 16-Core Processor
GPU(s)	NVIDIA GeForce RTX 3090