PHOTOGRAMMETRIE-EVALUATION

DANILO BARGEN, JOSUA SCHMID
COREDUMP RAPPERSWIL



3D-Rekonstruktion Schloss Rapperswil und Orthofoto-Erstellung HSR-Gelände mit Workflow, Software-Evaluation und Dokumentation

Technical Report Nr. 1501

coredump Rapperswil CH

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

INSTITUT FÜR SOFTWARE

Danilo Bargen, Josua Schmid: *Photogrammetrie-Evaluation: 3D-Rekonstruktion Schloss Rapperswil und Orthofoto-Erstellung HSR-Gelände mit Workflow, Software-Evaluation und Dokumentation.* © Fruehling–Sommer 2015

http://eprints.hsr.ch/id/eprint/439

AUFTRAGNEHMER:
Coredump Rapperswil

PROJEKTPARTNERIN:
HSR Hochschule fuer Technik Rapperswil
IFS Institut fuer Software / Geometa Lab
Prof. Stefan Keller

ORT:
Rapperswil, Schweiz

ZEITRAUM:
Fruehling–Sommer 2015

VERSION:
Version 1.0 2015-09-30

LIZENZ:
CC BY-SA 3.0 Switzerland

QUELLCODE DOKUMENTATION:
https://github.com/coredump-ch/photogrammetry-evaluation/

KONTAKT:
Danilo Bargen <mail@dbrgn.ch>
Josua Schmid <jschmid@fastmail.net>

ACKNOWLEDGEMENTS

Wir möchten uns bei folgenden Personen bedanken:

- Prof. Stefan Keller vom Geometa Lab der HSR für die tatkräftige Unterstützung bei der Realisation dieses Projektes
- Michael Suter von lightmoment.ch für das Erstellen der Drohnen-Luftbilder (Abschnitt 3.1 und Abschnitt 4.1)
- Jan Appenzeller von der Firma Drei-De für die Hilfe bei der Mesh-Nachbearbeitung (Abschnitt 3.6)
- Der Ortsgemeinde Rapperswil-Jona für das Bereitstellen von Grundrissen des Schlosses (Anhang D)
- Prof. Dr.-Ing. André Miede für das verwendete *classicthesis* LaTeX-Template

INHALTSVERZEICHNIS

Abbildungsverzeichnis viii
Glossar x

I EINLEITUNG 1

1 EINLEITUNG 3
1.1 Das Crowdfunding-Projekt . . . 3
1.2 Das 3D-Erfassungs-Projekt . . . 3
1.3 Ziele . . . 4

2 3D-REKONSTRUKTION 5
2.1 Rekonstruktion mittels Photogrammetrie . . . 5
2.2 Das Bildmaterial . . . 5
2.3 Feature-Erkennung . . . 5
2.4 Sparse Point Cloud . . . 5
2.5 Dense Point Cloud . . . 6
2.6 Mesh . . . 6
2.7 Solid . . . 6
2.8 Slicing . . . 7

II PRAXISBERICHT 9

3 3D-REKONSTRUKTION SCHLOSS RAPPERSWIL 11
3.1 Erfassung des Schlosses mit Kamera-Drohne . . . 11
3.1.1 Materialliste . . . 11
3.1.2 Vorgehen . . . 11
3.1.3 Bildqualität . . . 14
3.1.4 Learnings . . . 14
3.2 Nachbearbeiten des Bildmaterials . . . 15
3.3 Erster Versuch: Pix4D . . . 15
3.3.1 Learnings . . . 17
3.4 Zweiter Versuch: VisualSFM . . . 17
3.4.1 Installation . . . 17
3.4.2 Konfiguration . . . 18
3.4.3 Vorgehen . . . 18
3.5 Mesh-Generierung . . . 19
3.5.1 Mesh Öffnen . . . 20
3.5.2 Ebenen Kombinieren . . . 20
3.5.3 Punkte Löschen . . . 20
3.5.4 Mesh Rekonstruieren . . . 20
3.6 Mesh-Bereinigung . . . 22
3.6.1 Blender . . . 22
3.7 3D-Druck . . . 22
3.8 Resultate . . . 23

4 ERSTELLUNG ORTHOFOTOS UND DSM DER HSR 25

4.1 Erfassung des HSR-Geländes mit Kamera-Drohne 25
4.2 Geocoding der Bilder 27
4.3 Einführung OpenDroneMap 28
4.4 Installation OpenDroneMap 29
4.4.1 Lokale Installation 29
4.4.2 Vagrant / VirtualBox 29
4.4.3 Docker 29
4.5 Generierung Orthofoto und Oberflächenmodell 30
4.6 Resultate 31
4.6.1 Orthofoto 31
4.6.2 Texturiertes Oberflächenmodell 32
4.6.3 Beurteilung der Resultate 32
4.6.4 Learnings 33
4.6.5 Rohdaten 33
4.7 Weitere Ressourcen 34
4.7.1 Mapknitter 34
4.7.2 OpenAerialMaps 34

III SOFTWARE-EVALUATION 35

5 EINLEITUNG 37

6 3D REKONSTRUKTION 39
6.1 Wahl des Test-Objekts 39
6.2 Resultate 40
6.2.1 Rekonstruktionsdauer 40
6.2.2 Komplexität der Resultate 40

7 ERSTELLUNG ORTHOFOTO UND OBERFLÄCHENMODELL 47
7.1 Wahl des Test-Objekts 47
7.2 Resultate 48
7.2.1 Rekonstruktionsdauer 48
7.2.2 Komplexität der Resultate 48

IV APPENDIX 55

A APPENDIX A – FILESERVER / DOWNLOADS 57

B APPENDIX B – INSTALLATION VISUALSFM 59
B.1 Einleitung 59
B.1.1 Abhängigkeiten 59
B.1.2 Alternativen 60
B.2 Installation 60

C APPENDIX C – OPENDRONEMAP TIPPS 63
C.1 Kameraprofil erstellen 63

D APPENDIX D – GRUNDRISSE SCHLOSS RAPPERSWIL 65

E APPENDIX E – SOFTWARELISTE 69
E.1 Agisoft PhotoScan 69
E.2 Blender 69
E.3 CloudCompare 69
E.4 Cura 70

E.5 GPicSync 70
E.6 GPSBabel 70
E.7 Hugin 70
E.8 Menci APS 71
E.9 MeshLab 71
E.10 OpenDroneMap 71
E.11 Pix4Dmapper Pro 71
E.12 VisualSFM 72

LITERATURVERZEICHNIS 73

ABBILDUNGSVERZEICHNIS

Abbildung 1 GPS Trace des Quadrokopters während der Erfassung. Bildmaterial: © Bing Maps 12
Abbildung 2 GPS Trace des Quadrokopters während der Erfassung. Bildmaterial: © OpenStreetMap Contributors 12
Abbildung 3 Das Schloss Rapperswil von oben 13
Abbildung 4 Das Schloss Rapperswil von oben 13
Abbildung 5 Sparse Point Cloud in Pix4D 16
Abbildung 6 Thumbnails in VSFM 18
Abbildung 7 Sparse Point Cloud in VSFM 19
Abbildung 8 Dense Point Cloud in VSFM 19
Abbildung 9 Das Dialogfenster für die Poisson Reconstruction 21
Abbildung 10 Das aus der Poisson Reconstruction resultierende Mesh 21
Abbildung 11 Der Flatten/Contrast Brush 22
Abbildung 12 3D-Druck des Schlosses 23
Abbildung 13 3D-Druck des Schlosses 24
Abbildung 14 3D-Druck des Schlosses 24
Abbildung 15 Der Team BlackSheep Discovery Pro Quadrokopter 25
Abbildung 16 Die Kamera-Aufhängung 26
Abbildung 17 Deutlich sichtbar: Der gelbe GPS-Tracker . . . 26
Abbildung 18 GPS Trace des Quadkopters während der Erfassung. 27
Abbildung 19 Einstellungen GPicSync 28
Abbildung 20 Resultierendes Orthofoto des HSR-Areals . . . 31
Abbildung 21 Resultierendes Oberflächemodell des HSR-Areals 32
Abbildung 22 Evaluierte Photogrammetrie-Software: Lizenz / Kosten 37
Abbildung 23 Evaluierte Photogrammetrie-Software: Funktionsumfang 37
Abbildung 24 Erfassung Bildmaterial Strohhase 39
Abbildung 25 Resultat: Punktwolke Hase mit VSFM 41
Abbildung 26 Resultat: Punktwolke Hase mit Pix4DMapper Pro 42
Abbildung 27 Resultat: Punktwolke Hase mit PhotoScan Pro 43
Abbildung 28 Resultat: Mesh Hase mit Pix4DMapper Pro . . 44
Abbildung 29 Resultat: Mesh Hase mit PhotoScan Pro 45
Abbildung 30 Erfassung Bildmaterial HSR 47
Abbildung 31 Resultat: Orthofoto HSR mit OpenDroneMap . 49
Abbildung 32 Resultat: Orthofoto HSR mit Pix4Dmapper Pro 50

Abbildung 33 Resultat: Orthofoto HSR mit PhotoScan Pro . . 51
Abbildung 34 Resultat: Oberflächenmodell HSR mit OpenDroneMap 52
Abbildung 35 Resultat: Oberflächenmodell HSR mit Pix4Dmapper Pro 53
Abbildung 36 Resultat: Oberflächenmodell HSR mit PhotoScan Pro 54
Abbildung 37 Sensorgrösse SONY A5100 63
Abbildung 38 Schloss Rapperswil: Grundriss EG 65
Abbildung 39 Schloss Rapperswil: Grundriss 2. OG 66
Abbildung 40 Schloss Rapperswil: Grundriss 3. OG 67

GLOSSAR

DOM / DSM Digitales Oberflächenmodell (engl. Digital Surface Model)

G-Code Code bestehend aus CNC Instruktionen, wird häufig für 3D-Drucker einesetzt.

GIS Geografische Informationssysteme

GNSS Global Navigation Satellite System, z. B. GPS, GLONASS oder Galileo.

HSR Hochschule für Technik Rapperswil

Orthofoto Eine verzerrungsfreie und massstabsgetreue Abbildung der Erdoberfläche, die durch photogrammetrische Verfahren aus Luft- oder Satellitenbildern abgeleitet wird.

Photogrammetrie Eine Gruppe von Messmethoden und Auswerteverfahren der Fernerkundung, um aus Fotografien und genauen Messbildern eines Objektes seine räumliche Lage oder dreidimensionale Form zu bestimmen.

SFM Structure From Motion, eine Methode um 3D-Modelle aus «bewegten Bildern» zu erzeugen.

STL STereoLithography, ein für 3D-Druck geeignetes Dateiformat.

Teil I

EINLEITUNG

1 EINLEITUNG

Dieses Kapitel erläutert kurz den Ursprung dieses Projektes sowie dessen Ziele.

1.1 DAS CROWDFUNDING-PROJEKT

Wir sind Mitglieder des im Herbst 2013 in Rapperswil gegründeten Hackerspaces «Coredump»[1]. Unsere Ziele sind der kreative Umgang mit Technologie sowie der Know-How-Austausch an regelmässigen Treffen. Wir möchten unseren Mitgliedern gute Infrastruktur für technische Projekte bieten, v.a. in den Bereichen Informatik und Elektrotechnik.

Als Service für unsere Mitglieder wollten wir die Herstellung von selbst entworfenen Bauteilen mittels 3D-Drucker ermöglichen. Für dessen Finanzierung starteten wir im Januar 2015 eine Crowdfunding-Kampagne bei Wemakeit[2]. Wie es bei solchen Crowdfunding-Aktionen üblich ist, boten wir verschiedene Belohnungen für unsere Unterstützer an. Eine davon sollte lokalen Bezug haben. Wir entschieden uns daher dafür, das Schloss Rapperswil als 3D-Modell zu erfassen und in eine 3D-druckbare Form zu bringen.

Die Schweizer Crowdfunding-Plattform wemakeit wurde im Februar 2012 von der Kulturkommunikatorin Rea Eggli, dem Künstler Johannes Gees und dem Interaction Designer Jürg Lehni gegründet und ist mittlerweile die grösste Crowdfunding-Plattform in der Schweiz.

1.2 DAS 3D-ERFASSUNGS-PROJEKT

Zuerst wandten wir uns an die Ortsgemeinde Rapperswil-Jona, wo wir Grundrisse und weitere Architektur-Pläne des Schlosses erhielten (siehe Anhang D). Leider waren diese Pläne nicht vollständig genug, um daraus ein 3D-Modell des Schlosses zu rekonstruieren.

Da also keine Pläne verfügbar waren, mussten wir den 3D-Umriss des Schlosses selber erfassen. Unser erster Gedanke war die Vermessung mittels Laserscanning. Dafür wandten wir uns an Prof. Stefan Keller an der HSR, um von seinem Know-How im Bereich Geowissenschaft / Geomatik / GIS zu profitieren.

Prof. Keller schlug uns jedoch vor, das Schloss stattdessen mithilfe einer ferngesteuerten Kamera-Drohne zu erfassen und mit Photogrammetrie-Software weiterzuverarbeiten. Wir begannen mit der Recherche-Arbeit in diesem Themenfeld und so entstand ein Projektplan. Die HSR würde die Finanzierung übernehmen, wir bieten dem Geometa Lab der HSR dafür im Gegenzug das gesammelte Know-How in Form dieser Dokumentation.

1 https://www.coredump.ch/
2 https://wemakeit.com/projects/3d-drucker-fuer-rapperswil/

1.3 ZIELE

Die Ziele dieses Projektes wurden wie folgt definiert:

1. Erarbeitung eines Workflows zur Erstellung eines 3D-Modells des Schloss Rapperswils aus Drohnen-Luftbildern, bevorzugt mithilfe von Open Source Software.
2. Evaluation von verschiedenen Software-Lösungen zur Erstellung einer 3D-Figur aus einem physischen Objekt mittels Photogrammetrie.
3. Evaluation von verschiedenen Software-Lösungen zur Erstellung eines Orthophotos (entzerrtes 2D-Luftbild) aus Drohnen-Luftbildern.

Diese Dokumentation ist das Ergebnis und erfüllt alle obenstehenden Ziele.

2 3D-REKONSTRUKTION

2.1 REKONSTRUKTION MITTELS PHOTOGRAMMETRIE

Es gibt verschiedene Methoden, um ein Objekt aus der echten Welt in ein digitales 3D-Modell umzuwandeln. Eine davon ist die Photogrammetrie.

Der Begriff Photogrammetrie beschreibt Methoden, um aus zweidimensionalen Bildern ein dreidimensionales Objekt zu rekonstruieren.

In unserem Fall möchten wir nicht beim 3D-Modell aufhören, sondern dieses in ein für 3D-Drucker geeignetes Format bringen. Wir möchten also ein physikalisches Objekt digitalisieren und anschliessend wieder mit einem 3D-Drucker ausdrucken.

2.2 DAS BILDMATERIAL

Die ideale Ausgangsquelle für 3D-Rekonstruktion mittels Photogrammetrie ist eine hohe Anzahl qualitativ hochwertiger Fotos des Zielobjektes von allen Seiten. Jede Oberfläche sollte darin zu sehen sein. Die Bilder sollten sich überlappen, damit die Photogrammetrie-Software daraus die ursprüngliche Oberfläche errechnen kann. Idealerweise enthalten die Bilder auch GNSS-Koordinaten. Damit ist eine einfache Georeferenzierung möglich und der Rekonstruktions-Prozess kann beschleunigt werden.

Bei der Georeferenzierung werden Daten in Bezug zu geografischen Gegebenheiten gesetzt. So können Skalierungs-Faktoren oder Koordinaten einzelner Datenpunkte im Raum ermittelt werden. Ein mögliches Beispiel dafür ist ein Luftbild, welches passgenau über eine Karte gelegt wird.

2.3 FEATURE-ERKENNUNG

Als nächster Schritt wird das Bildmaterial in eine Photogrammetrie-Software geladen. Diese versucht nun, auf den Bildern sogenannte «Features» zu erkennen. Features (Deutsch: Merkmale) sind mathematisch interessante Stellen eines Bildes, welche genutzt werden können, um die Dimensionen eines Objektes unabhängig von dessen Skalierung oder Verzerrung zu extrahieren.

Die Features sind eine wichtige Grundlage für die weitere Verarbeitung im Rekonstruktions-Prozess.

2.4 SPARSE POINT CLOUD

Die dünn besetzte Punktwolke, im Englischen «Sparse Point Cloud» genannt, besteht aus Punkten im 3D-Raum. Sie enthält die nötigsten Punkte aus der 3D-Rekonstruktion und lässt bereits die Konturen

des Objektes erahnen. Sie stellt jedoch noch keine vollständige Rekonstruktion des Objektes dar.

Die Punktwolke wird aus den im Schritt 2.3 erkannten Features generiert. Ein Feature, das in mehreren Bildern erkannt wird, kann als gemeinsamer Referenzpunkt im 3D-Raum genutzt werden und wird dann zu einem Punkt in der Punktwolke.

2.5 DENSE POINT CLOUD

Nachdem die dünn besetzte Punktwolke generiert wurde, kann nun der leere Raum zwischen den 3D-Punkten mithilfe des existierenden Bildmaterials und den Referenzpunkten eingefüllt werden. Dieser Prozess nennt sich auch «Verdichtung» oder «Densification». Als Ergebnis erhält man eine dicht besetzte Punktwolke, die ein Vielfaches der Punkte aus der dünn besetzten Punktwolke enthält. Die Punkte können zudem Metadaten wie beispielsweise Farbinformationen speichern.

2.6 MESH

Aus der dicht besetzten Punktwolke kann nun ein Objekt mit einer Oberfläche, das sogenannte «Mesh», generiert werden. Ein Mesh, auch Polygonnetz genannt, ist eine Oberfläche bestehend aus vielen aneinanderhängenden Polygonen, in der Regel Drei- oder Vierecken.

Die Punktwolke enthält in der Regel zu viele und auch falsche Punkte. Um daraus ein Mesh zu erzeugen, muss die Punktwolke zuerst bereinigt werden. Dies geschieht, indem nicht benötigte Punkte gelöscht und dichte Punktgruppen auf einen Punkt zusammengefasst werden. Diese Rausch-Reduzierung verhilft den Mesh-Rekonstruktions-Algorithmen zu besseren Ergebnissen. In unserem Fall wurde die Bereinigung von Hand durchgeführt.

Zur Umwandlung einer bereinigten Punktwolke in ein Mesh existieren ebenfalls diverse Algorithmen. Ein besonders gut geeigneter Algorithmus ist der von Microsoft Research im Jahr 2006 entwickelte «Poisson Surface Reconstruction»[6] Algorithmus.

2.7 SOLID

Damit ein Mesh druckbar wird, muss es in ein solides Objekt umgewandelt werden. Ein Mesh besteht nur aus einer Oberfläche, ein solides Objekt hinegegen ist «wasserdicht» und hat ein Volumen. Wichtig bei der Umwandlung ist auch, dass das Mesh eine mathematische Mannigfaltigkeit (Englisch: Manifold) ist. Das bedeutet unter anderem, dass alle Flächen nach aussen zeigen, dass keine Vertizes oder Kanten mehrfach vorhanden sind und dass an jeder Kante zwei Flächen anliegen.

Sind diese Voraussetzungen gegeben, kann ein Mesh mit geeigneter Software in ein Solid-Format wie STL[1] exportiert werden.

2.8 SLICING

Damit ein solides Objekt von einem 3D-Drucker gedruckt werden kann, muss es bei in druckbare Schichten umgewandelt werden. Dieser Prozess nennt sich «Slicing» und wird in der Regel von der dem 3D-Drucker mitgelieferten Software erledigt. Das nahezu universell dafür benutzte Datenformat ist der sogenannte G-Code[2].

G-Code ist ein Steuerungscode für computergesteuerte Maschinen wie CNC-Fräsen oder 3D-Drucker. Er enthält Befehle und Koordinaten um den Druckkopf zu positionieren und das Material zu extrudieren. Auch die Temperatur, die Lüfter und weitere Parameter können angesteuert werden.

Beim Slicing werden ideale Bahnen für den Druckkopf berechnet. Dabei werden Parameter wie Schichtdicke, Druckkopf-Durchmesser, Geschwindigkeit usw. berücksichtigt. Die meisten Slicer ermöglichen auch die Generierung von Stützmaterial oder Haftungshilfen wie einem «Raft» oder «Brim».

Der resultierende G-Code kann dann direkt an den 3D-Drucker gesendet werden. Der Rekonstruktions-Workflow ist nun abgeschlossen, das gescannte Objekt entsteht nun auf der Druckplatte.

1 STereoLithography, https://de.wikipedia.org/wiki/STL-Schnittstelle
2 https://en.wikipedia.org/wiki/G-code

Teil II

PRAXISBERICHT

3

3D-REKONSTRUKTION SCHLOSS RAPPERSWIL

Die folgenden Kapitel beschreiben den Prozess, der nötig war um das Schloss Rapperswil mittels Luftbilder als 3D-Modell zu rekonstruieren. Wir gehen hier nicht auf die Evaluation der genutzten Software ein, dies folgt im nächsten Teil dieser Dokumentation.

Die Aufnahmen des Schlosses wurden am 28. Januar 2015 erstellt.

3.1 ERFASSUNG DES SCHLOSSES MIT KAMERA-DROHNE

Um Fotos des Schlosses von allen Winkeln zu erstellen, braucht man ein ferngesteuertes Flugobjekt, wie z. B. einen Multikopter. In unserem Fall wurden die Fotos von Michael Suter (http://lightmoment.ch/) erstellt, der einen Quadrokopter besitzt und damit freundlicherweise seine Piloten-Fähigkeiten unter Beweis gestellt hat.

3.1.1 *Materialliste*

- Quadrokopter: *Team BlackSheep Discovery Pro*[1]
- Kamera: *GoPro Hero 4 Black*[2]
- GPS Tracker: *Fairphone FP1*[3] mit *GeoTracker* App[4]

3.1.2 *Vorgehen*

Die maximale Flugzeit pro Akku beträgt 10–15 Minuten. Wir hatten zwei geladene Akkus dabei und erreichten somit eine Flugzeit von 20–30 Minuten.

Die GoPro Kamera hatten wir so eingestellt, dass sie zwei mal pro Sekunde ein Bild machte. Daraus ergaben sich dann etwa 2700 Fotos, was ca. 5.3 GiB Bildmaterial entspricht.

Um Bewegungs-Unschärfe bei den Bildern zu vermeiden, ist es wichtig, dass der Multikopter über eine Kamera-Stabilisierung verfügt. Dies ist bei der genutzten TBS Discovery mit einem Gimbal (gyroskopische Mehrachsen-Stabilisierung) gegeben.

Zur Aufzeichnung der GPS-Koordinaten haben wir ein Smartphone mit einer GPS Tracking App auf dem Quadrokopter befestigt.

1 http://www.team-blacksheep.com/products/prod:discopro
2 https://gopro.com/
3 https://www.fairphone.com/
4 https://play.google.com/store/apps/details?id=com.ilyabogdanovich.geotracker

Mit dem Quadrokopter sind wir dann während einer halben Stunde mehrmals um das Schloss herumgeflogen um Fotos von jedem Winkel zu erstellen. Auch den Innenhof haben wir mit dem Quadrokopter abgeflogen. Anschliessend sind wir noch zu Fuss mit der Kamera in der Hand dem Pfad neben dem Schloss gefolgt um auch gutes Bildmaterial innerhalb des Tores zu erhalten.



Abbildung 1: GPS Trace des Quadrokopters während der Erfassung.
Bildmaterial: © Bing Maps

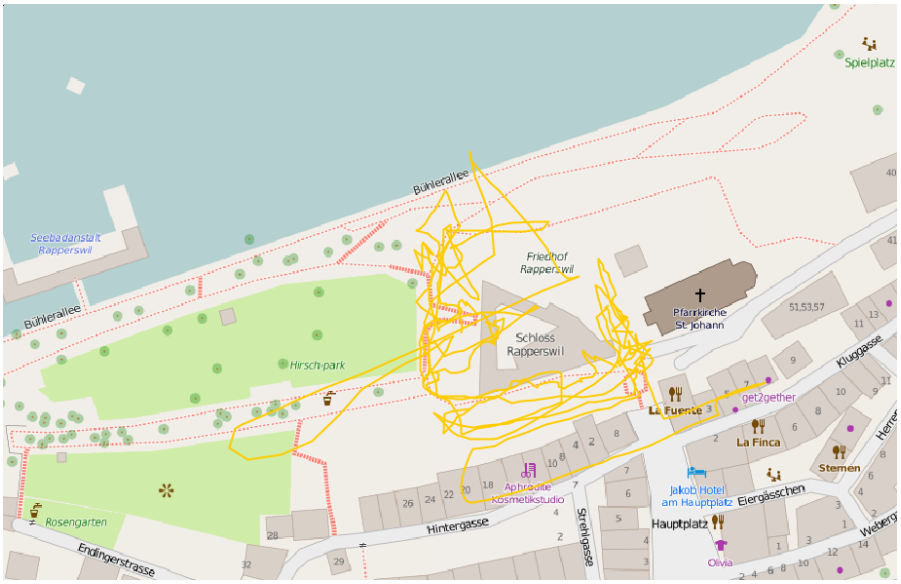


Abbildung 2: GPS Trace des Quadrokopters während der Erfassung.
Bildmaterial: © OpenStreetMap Contributors

Ein Video der 3D-Erfassung ist unter https://vimeo.com/118125082 verfügbar. Nachfolgend noch zwei Aufnahmen des Quadrokopters:



Abbildung 3: Das Schloss Rapperswil von oben



Abbildung 4: Das Schloss Rapperswil von oben

3.1.3 *Bildqualität*

Die GoPro Kamera ist für Sportaktivitäten entwickelt worden und hat daher ein Weitwinkel-Objektiv («Fisheye») eingebaut. Dies ist bestens geeignet um beispielsweise beim Skifahren einen guten Überblick zu bewahren, für die Rekonstruktion ist es aber eher nachteilig.

Ein weiterer Faktor ist die Kompression. Um viele Bilder auf der SD-Karte speichern zu können, werden die JPEG-Bilder stark komprimiert. Die daraus entstehenden JPEG-Artefakte können die Feature-Detection stören.

Das Raw-Format, auch Rohdatenformat genannt, bezeichnet eine Familie von Dateiformaten für Digitalkameras, bei denen die Kamera die Bildinformationen nach der Digitalisierung direkt ohne weitere Bearbeitung auf das Speichermedium schreibt. Bilder im Rohdatenformat werden gelegentlich auch als «Digitales Negativ» bezeichnet.

Idealerweise würde man daher zur Erfassung eine Spiegelreflex-Kamera mit neutralem 35mm Objektiv verwenden und alle Bilder im Raw-Format abspeichern. Diese können dann mit entsprechender Software ohne verlustbehaftete Kompression in ein für die Photogrammetrie-Software nutzbares Format umgewandelt werden.

In unserem Fall waren wir aber durch die Tragfähigkeit des Quadrokopters eingeschränkt und haben uns stattdessen dafür entschieden, die Bilder am Computer mithilfe eines Linsenprofils in Photoshop Lightroom[5] zu entzerren.

Inzwischen gibt es zwar Photogrammetrie-Programme welche die GoPro Linsenprofile direkt unterstützen. Nichtsdestotrotz gibt es sicher Kameras, welche besser für solche Aufgaben geeignet sind als die GoPro.

3.1.4 *Learnings*

- Die Jahreszeit war an sich gut gewählt, da die Bäume im Winter nicht belaubt sind, wodurch die Sicht auf das Schloss nicht eingeschränkt wird.
- Nachteilig war jedoch der Schnee auf den Dächern. Er überdeckte die Dachziegel und führte so bei der Feature-Erkennung zu schlechteren Ergebnissen.
- Die GoPro Kamera hat eine äusserst starke Linsen-Verzerrung. Eine Kamera mit neutraler Linse wäre vermutlich besser geeignet.
- Die JPEG Kompression ist möglicherweise störend. Eine Kamera mit Unterstützung für unkomprimierte Bilder (Raw-Format) würde vielleicht bessere Resultate liefern.

5 http://www.photoshop.com/products/photoshoplightroom

3.2 NACHBEARBEITEN DES BILDMATERIALS

Um wie im letzten Kapitel bereits besprochen, haben wir unsere Bilder mithilfe von Photoshop Lightroom nachbearbeitet um die Linsenverzerrung herauszurechnen.

Das Linsenprofil der GoPro 4 wird seit Photoshop Lightroom 5.7 unterstützt. Die Korrektur kann für alle Bilder gleichzeitig im Batch-Korrektur-Modus erfolgen. Nach der Korrektur haben wir die Bilder mit JPEG-Qualitätsstufe 100 exportiert.

Als Open Source Alternative zu Photoshop Lightroom wäre noch Hugin[6] zu erwähnen, mit dem ebenfalls Bilder entzerrt werden können. Wir haben jedoch kein vorbereitetes Linsenprofil für die GoPro 4 gefunden und uns deshalb aus Zeitgründen für Lightroom entschieden. Ein Linsenprofil könnte jedoch selber erstellt werden. Anleitungen dazu gibt es online[7]. Korrektur-Profile für die GoPro 3+ Silver können unter `https://github.com/racerxdl/goprocorrect` gefunden werden.

3.3 ERSTER VERSUCH: PIX4D

Die erste von uns getestete Software war Pix4D[8]. Dieses Programm wurde von einem EPFL-Spinoff in der Westschweiz entwickelt. Während die Vollversion relativ teuer ist (7'900 CHF), gibt es eine Testversion mit eingeschränktem Funktionsumfang (keine GPU-Beschleunigung, kein Export), welche wir für unseren ersten Versuch genutzt haben.

Da die Software damit wirbt, problemlos mit grossen Datenmengen umgehen zu können, haben wir direkt mal alle 2700 Fotos hineingeladen und einen Rekonstruktions-Prozess gestartet. (Da Pix4D direkt Unterstützung für die GoPro Linse mitbringt, ist die Entzerrung übrigens optional. Wir haben in diesem Schritt darauf verzichtet.)

Die Rekonstruktion dauerte auf einem Intel Core i7-4790K mit 16 GiB RAM etwas über 52 Stunden. Das resultierende Modell war jedoch leider fehlerhaft. Wie man auf der Abbildung 5 erkennen kann, wurde die Gasse neben dem Schloss der Schlossmauer entlang schräg nach oben geführt. Zudem war im endgültigen Ergebnis der Hauptturm doppelt vorhanden, einmal schräg versetzt.

Unsere Vermutung ist, dass wir zu viele (teilweise qualitativ nicht einwandfreie) Input-Daten verwendet hatten, so dass sich falsche Matches so weit summiert haben, dass die Software sie als gültig betrachtet hat. Wir konnten diese Hypothese aus Zeitgründen jedoch nicht verifizieren.

6 http://hugin.sourceforge.net/
7 http://hugin.sourceforge.net/tutorials/calibration/en.shtml
8 https://pix4d.com/

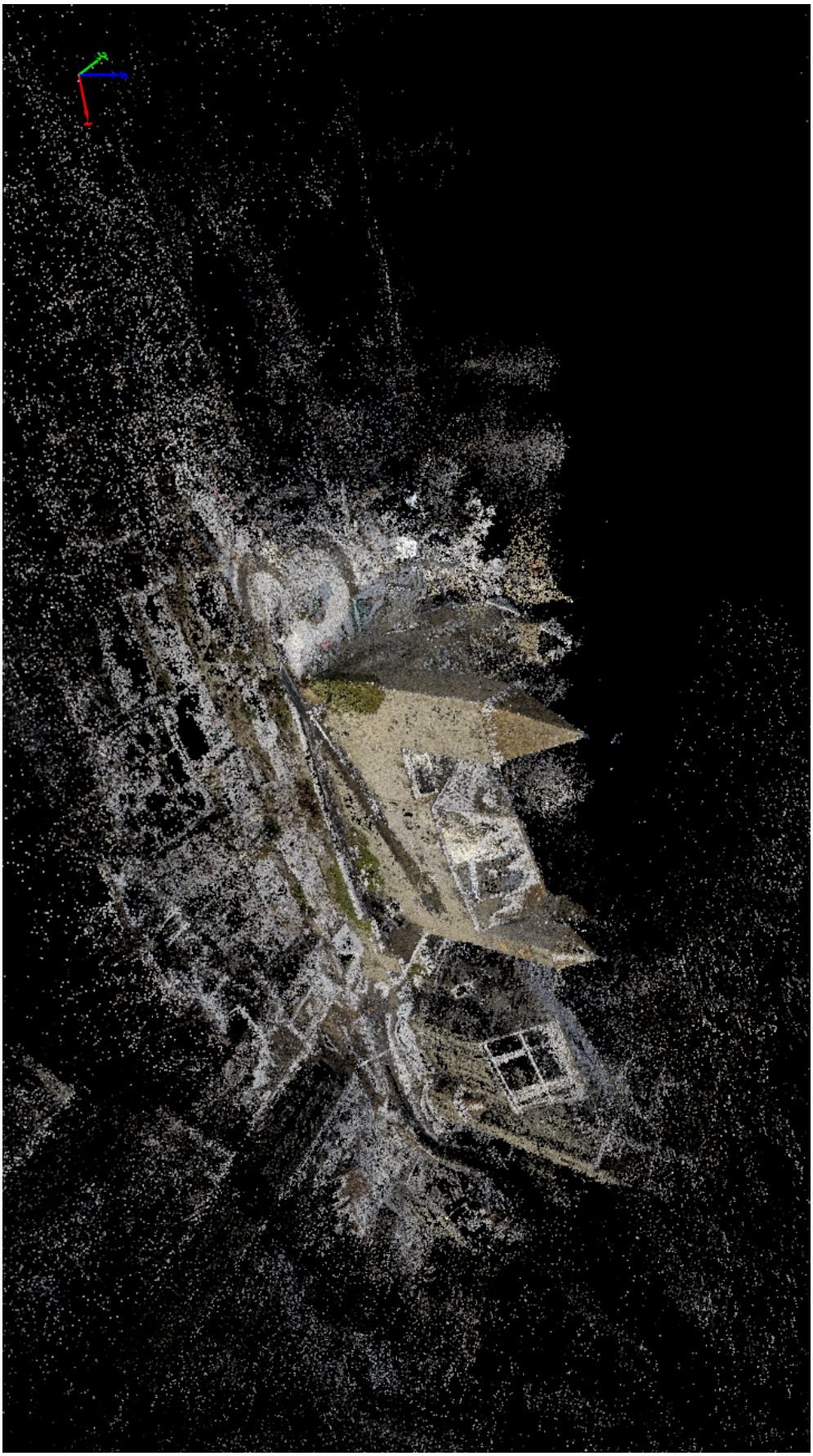


Abbildung 5: Sparse Point Cloud in Pix4D

3.3.1 *Learnings*

- Vermutlich ist es besser, die Anzahl Bilder auf die qualitativ hochwertigen zu reduzieren. Das senkt die Chance für fehlerhafte Matches.
- Ein Rekonstruktions-Algorithmus mit Unterstützung für sequentielle Bilder (oder Videos) könnte möglicherweise viel bessere Resultate liefern, da wiederum die Chance von falschen Matches reduziert wird.
- Möglicherweise war der zeitliche Abstand zwischen den Einzelbildern zu gross. Bei der Analyse zeitnaher Bilder würden wichtige Features stärker gewichtet, was eine Point-Cloud mit prozentual weniger falschen Punkten ergäbe.

3.4 ZWEITER VERSUCH: VISUALSFM

Nach dem ersten Versuch mit Pix4D haben wir einen zweiten Versuch mit VisualSFM[9][13][11] unter Linux gestartet. VisualSFM (Visual Structure From Motion, abgekürzt VSFM) wurde von Changchang Wu im Jahr 2011 im Rahmen seines Postdoc Researches als Frontend für die Photogrammetrie-Tools SiftGPU[12], Multicore Bundle Adjustment[14], CMVS[4], PMVS[3], CMPMVS[5], MVE[10], SURE[9], MeshRecon (und weitere) entwickelt. Der Quellcode ist quelloffen, allerdings darf die Software gemäss ihrer proprietären Lizenz ohne Kontaktaufnahme mit dem Autor nur zu akademischen und nonkommerziellen Zwecken genutzt werden. Die verwendeten Komponenten sind teilweise unter einer freien Lizenz[10] veröffentlicht, teilweise wie VSFM selbst nur für akademische und nonkommerzielle Zwecke nutzbar.

3.4.1 *Installation*

Die Installation von VSFM kann etwas herausfordernd sein, deshalb haben wir im Anhang B die Installation unter Ubuntu 14.04 beschrieben. Alternativ ist unter Arch Linux die Installation dank Paketen im Arch User Repository[11] relativ einfach möglich.

Auf `https://registry.hub.docker.com/u/ryanfb/visualsfm/` haben wir zudem auch noch ein Docker-Image für VSFM gefunden. Angesichts der komplexen Installation ist eine Containerisierung sicherlich eine gute Idee. Zum Zeitpunkt dieser Dokumentation lief das Image auf unserer Testmaschine jedoch noch nicht fehlerfrei.

9 `http://ccwu.me/vsfm/`
10 `https://www.fsf.org/licensing/`
11 `https://aur.archlinux.org/packages/vsfm/`

3.4.2 *Konfiguration*

VSFM verfügt über viele versteckte Konfigurationsoptionen in der `nv.ini` Konfigurationsdatei. Diese editiert man am einfachsten, indem man die `CTRL`-Taste gedrückt hält, während man auf das Settings-Icon klickt. Nun sollte sich ein Texteditor öffnen. Für einen ersten Versuch sollte jedoch die Standard-Konfiguration ausreichen.

Für schnellere Resultate kann man optional die CUDA-Unterstützung unter `Tools > Enable GPU > Match using CUDA` aktivieren.

3.4.3 *Vorgehen*

Beim Vorgehen haben wir uns an einem gut gemachten Tutorial auf YouTube orientiert: `https://youtu.be/V4iBb_j6k_g`

Zuerst haben wir wie im Abschnitt 3.2 beschrieben die Fotos einer Linsenkorrektur unterzogen. Anschliessend haben wir 571 Bilder für einen Testlauf ausgewählt und in VSFM geladen. Bei der Auswahl haben wir darauf geachtet, dass jeder Teil des Schlosses darin mehrfach vertreten ist.

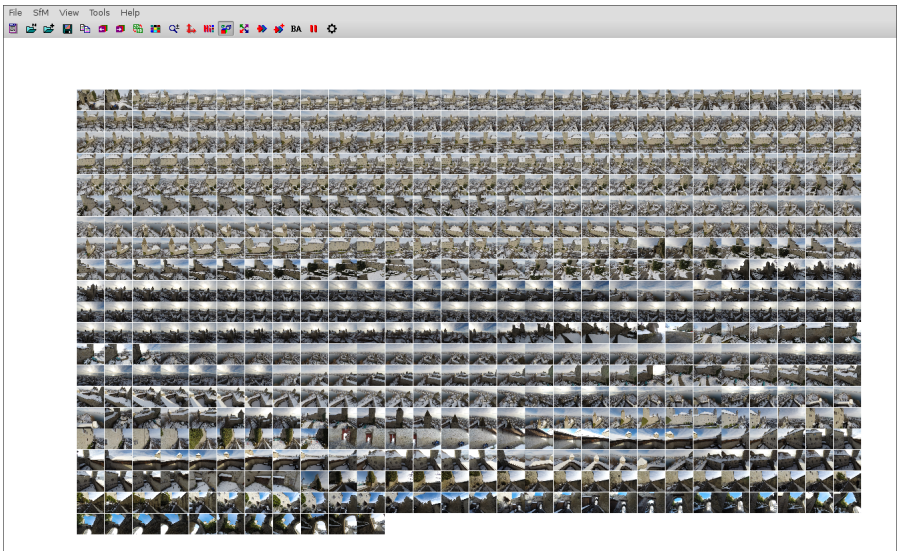


Abbildung 6: Thumbnails in VSFM

Mit einem Klick auf «Compute Missing Matches» wird die Feature-Erkennung gestartet.

Wenn diese fertig ist, kann mit einem Klick auf «Compute 3D Reconstruction» die Rekonstruktion der Sparse Point Cloud (siehe Abschnitt 2.4) beginnen. Man kann diese in Echtzeit überwachen. Das Resultat lässt bereits die Umrisse des Schlosses erahnen.

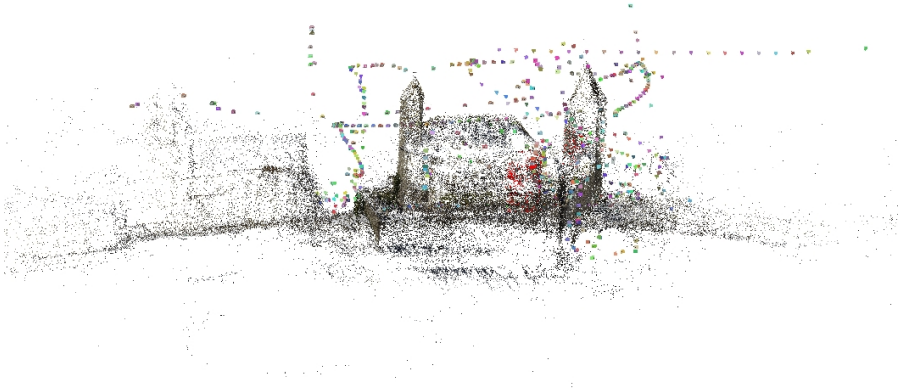


Abbildung 7: Sparse Point Cloud in VSFM

Als nächstes folgt die Schwerarbeit: Die Verdichtung der Punktwolke (siehe Abschnitt 2.5). Dieser Prozess kann mit einem Klick auf «Run Dense Reconstruction» gestartet werden. Er dauert mehrere Stunden, das Resultat sah folgendermassen aus:



Abbildung 8: Dense Point Cloud in VSFM

3.5 MESH-GENERIERUNG

Um ein Mesh zu generieren, muss die Punktwolke zuerst bereinigt werden. Wie man in der Abbildung 8 sehen kann, gibt es um das Schloss herum noch viele Artefakte, welche man zuerst löschen muss.

Wir haben für diesen Prozess MeshLab[12] verwendet. MeshLab ist eine Software unter der GPL Lizenz, welches zur Ver- und Bearbeitung von Meshes benutzt werden kann. Das Programm ist etwas gewöhnungsbedürftig und lässt in der Bedienung zu Wünschen übrig, die eingebauten Funktionen sind jedoch sehr mächtig.

12 http://meshlab.sourceforge.net/

3.5.1 *Mesh Öffnen*

Die ersten Probleme tauchten beim Laden der Punktwolke auf. Der Memory-Verbrauch von Meshlab ist riesig. Um die 4.5 GiB grosse Punktwolke zu laden, wurden 29 GiB RAM gefüllt. Auf unserem Testgerät waren zuerst nur 16 GiB RAM + 8 GiB Swapspace vorhanden, was dazu führt, dass MeshLab crasht sobald kein Speicher mehr verfügbar ist. Durch Erstellen eines grösseren Swapfiles konnte das Problem umgangen werden, die Aufrüstung von RAM ist jedoch die bessere Lösung.

3.5.2 *Ebenen Kombinieren*

Wenn die Punktwolke in MeshLab geöffnet ist, müssen zuerst die Ebenen kombiniert werden. VSFM erstellt die Rekonstruktion in mehreren Clustern, welche auch in eigenen `.ply`-Dateien abgelegt werden. Diese Cluster werden in MeshLab als Ebenen geladen. Die Software ermöglicht es leider nicht, Punkte aus mehreren Ebenen gleichzeitig zu selektieren, um sie zu löschen. Deshalb müssen die vielen Ebenen auf eine einzigen reduziert werden.

Zuerst muss mit `Ctrl+L` der Layer-Dialog eingeblendet werden. Danach kann man mit einem Rechtsklick auf eine beliebigen Ebene die Aktion «Flatten Visible Layers» auswählen.

Achtung: Beim Kombinieren der Ebenen geht die Textur-Information verloren. In unserem Fall war das egal, falls die Texturen erhalten werden müssen, gibt es bestimmt eine Lösung für dieses Problem.

3.5.3 *Punkte Löschen*

Nun kann man die überflüssigen Punkte in der Punktwolke löschen. Dafür klickt man in der Toolbar zuerst auf «Select Vertexes»  , dann zeichnet man einen rechteckigen Bereich über die Punktwolke. Die selektieren Punkte können dann mit «Delete the current set of selected vertices»  gelöscht werden.

Ein wichtiger Punkt welcher uns in MeshLab aufgefallen ist: Es gibt keine Undo-Funktion! Wenn ein Punkt gelöscht wird, ist er unwiderruflich weg. Deshalb sollten beim Bearbeiten von Punktwolken und Meshes alle paar Minuten Backups erstellt werden.

3.5.4 *Mesh Rekonstruieren*

Das Mesh kann nun aus der Punktwolke rekonstruiert werden. MeshLab beinhaltet dafür diverse Algorithmen, ein besonders gut geeigneter Algorithmus ist der von Microsoft Research im Jahr 2006 entwickelte «Poisson Surface Reconstruction»[6] Algorithmus.

Dieser kann über das Menu `Filters > Remeshing, Simplification and Reconstruction > Surface Reconstruction: Poisson` gestartet werden. Nun öffnet sich ein Dialogfenster:

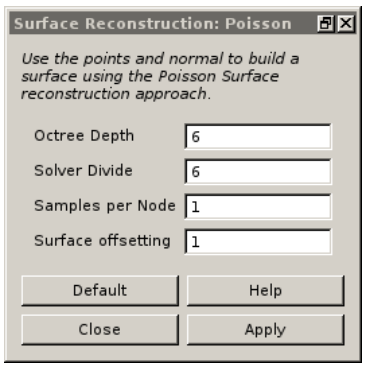


Abbildung 9: Das Dialogfenster für die Poisson Reconstruction

Durch das Anpassen der Parameter kann das Resultat der Rekonstruktion beeinflusst werden. Wir haben durch Experimentieren zwei verschiedene relativ gut funktionierende Einstellungen gefunden:

- `12, 10, 1, 1`: Viele Details sichtbar
- `12, 8, 10, 1`: Mehr Glättung, weniger Störungen

Der dritte Parameter, «Samples per Node», steht dafür, wie viele Punkte der Punktwolke verwendet werden, um einen Vertex im Mesh zu erstellen. Das erklärt die Glättung, die damit ersichtlich ist.

Hier das Resultat der Rekonstruktion:

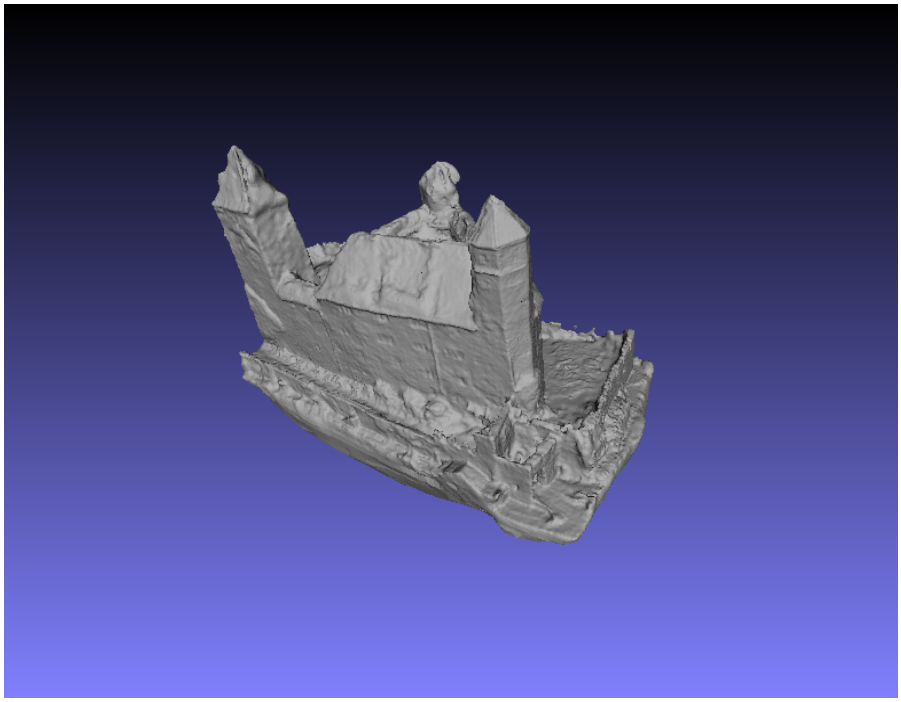


Abbildung 10: Das aus der Poisson Reconstruction resultierende Mesh

3.6 MESH-BEREINIGUNG

Das Mesh ist nun bereit, sieht allerdings noch roh aus und weist viele Unschönheiten auf. Diese müssen nun korrigiert werden. In unserem Fall geschah dies von Hand.

Bei diesem Prozess wurden wir aus Zeitgründen zu Beginn von der Firma Drei-De in Rapperswil unterstützt. Sie übernahmen das initiale Mesh-Rework.

Anschliessend haben wir die letzten Korrekturen mit der Software Blender[13] durchgeführt.

3.6.1 *Blender*

Blender ist ebenfalls freie Software unter der GPL Lizenz. Das Programm kann genutzt werden, um dreidimensionale Körper zu modellieren, sie zu texturieren, zu animieren und zu rendern. Mit Hilfe von Blender wurden bisher schon mehrere Kurzfilme produziert, darunter Sintel (2010, [1]), Tears of Steel (2012, [2]) und weitere[14].

Wir haben primär mit den Modellier-Werkzeugen im sogenannten «Sculpt Mode» von Blender gearbeitet. Diese sind der Arbeit mit Modelliermasse nachempfunden. Man kann beispielsweise rauhe Stellen glätten, hervorstehende Flächen «hineindrücken», etc.

In unserem Fall mussten wir vor allem unebene Flächen glätten. Dies kann man mit dem «Flatten/Contrast» Brush bei gedrückter `Shift`-Taste einfach erledigen.

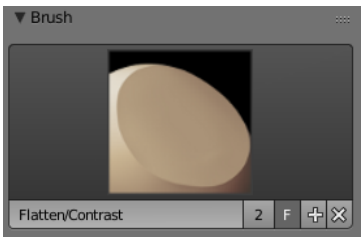


Abbildung 11: Der Flatten/Contrast Brush

Das fertige Mesh kann nun unter `File > Export > Stl (.stl)` ins STL Format exportiert werden.

3.7 3D-DRUCK

Für das Slicing (Abschnitt 2.8) des Meshes in ein Modell mit ca. 10 cm Kantenlänge haben wir Cura[15] mit folgenden Einstellungen verwendet:

13 https://www.blender.org/
14 http://archive.blender.org/features-gallery/movies/
15 https://ultimaker.com/en/products/cura-software

- Layer Height: 0.1mm
- Shell Thickness: 0.8mm
- Bottom / Top Thickness: 1.1mm
- Fill Density: 20%
- Print Speed: 50mm/s
- Support Type: None
- Platform Adhesion Type: Brim

Damit haben wir mit dem Ultimaker 2 relativ gute Erfahrungen gemacht. Der Druckvorgang dauert dabei ca. 11 Stunden.

3.8 RESULTATE

Das fertig druckbare 3D-Modell kann unter `https://www.thingiverse.com/thing:734125` heruntergeladen werden. Ein Video des Druckvorganges ist hier verfügbar: `https://vimeo.com/122413144`

Alle Rohdaten können vom Coredump Fileserver heruntergeladen werden. Weitere Infos im Anhang A.

Nachfolgend noch ein paar Fotos der 3D-gedruckten Resultate:



Abbildung 12: 3D-Druck des Schlosses

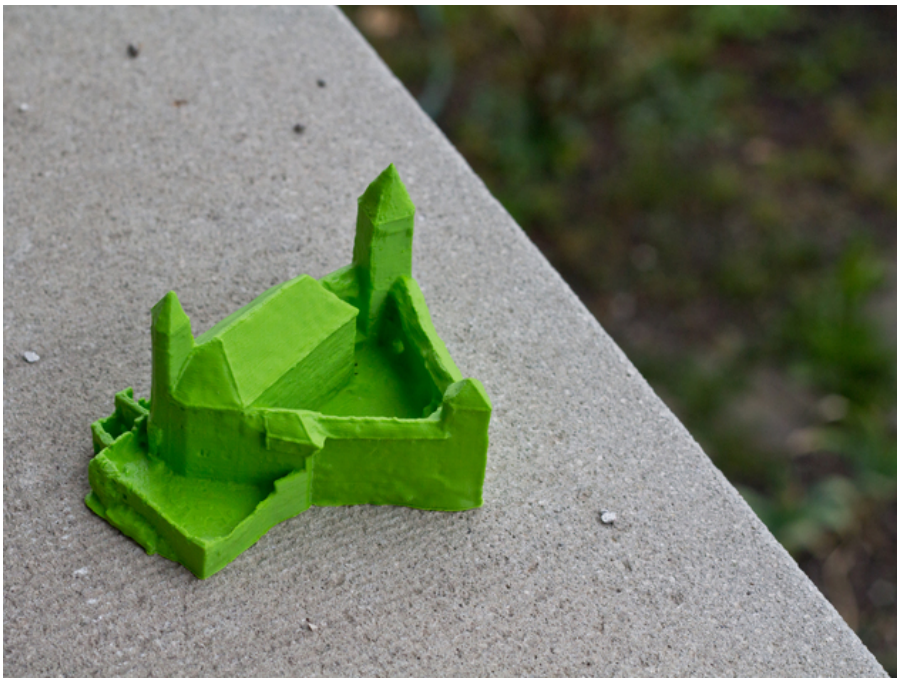


Abbildung 13: 3D-Druck des Schlosses



Abbildung 14: 3D-Druck des Schlosses

4

ERSTELLUNG ORTHOFOTOS UND DSM DER HSR

Die folgenden Kapitel beschreiben den Prozess, der nötig war um ein Orthofoto sowie ein georeferenziertes texturiertes Oberflächenmodell der Hochschule für Technik in Rapperswil zu erzeugen.

Die Aufnahmen des HSR Geländes wurden am 8. August 2015 erstellt.

4.1 ERFASSUNG DES HSR-GELÄNDES MIT KAMERA-DROHNE

Auf die Erfassung von Luftbildern mithilfe eines Multikopters sind wir bereits in Abschnitt 3.1 eingegangen. Deshalb möchten wir hier lediglich die Unterschiede zur 3D-Erfassung des Schloss Rapperswils hervorheben.

Im Gegensatz zur 3D-Erfassung geht es bei der Herstellung von Orthofotos nicht primär um die dreidimensionale Form eines Objektes, sondern darum, möglichst senkrechte und verzerrungsfreie Bilder zu erstellen. Der Flugplan war daher, in ca. 100m Höhe mehrmals über die HSR zu fliegen und alle paar Meter ein Foto zu erstellen, so dass die gesamte Zielfläche abgelichtet wurde.

Wir haben wiederum den Team BlackSheep Discovery Pro Quadrokopter mit Gimbal verwendet, diesmal jedoch nicht mit der GoPro Kamera, sondern mit einer Sony Alpha 5100 und einer verzerrungsfreien Brennweite.



Abbildung 15: Der Team BlackSheep Discovery Pro Quadrokopter

Die Kamera wurde mit dem Gimbal so positioniert, dass sie senkrecht nach unten zeigt.

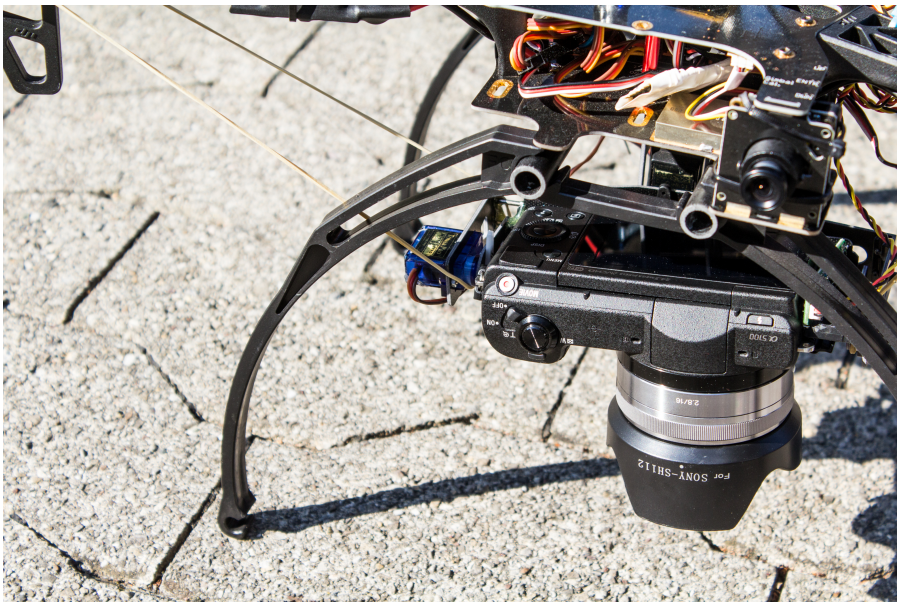


Abbildung 16: Die Kamera-Aufhängung

Während wir beim ersten Versuch ein Smartphone als GPS-Tracker auf dem Quadrokopter befestigt hatten, verwendeten wir diesmal ein dediziertes GPS-Tracking-Gerät, ein *Navin miniHomer*[1]. Der Tracker wurde so konfiguriert, dass er 10 mal pro Sekunde die Position und Ausrichtung aufzeichnete.

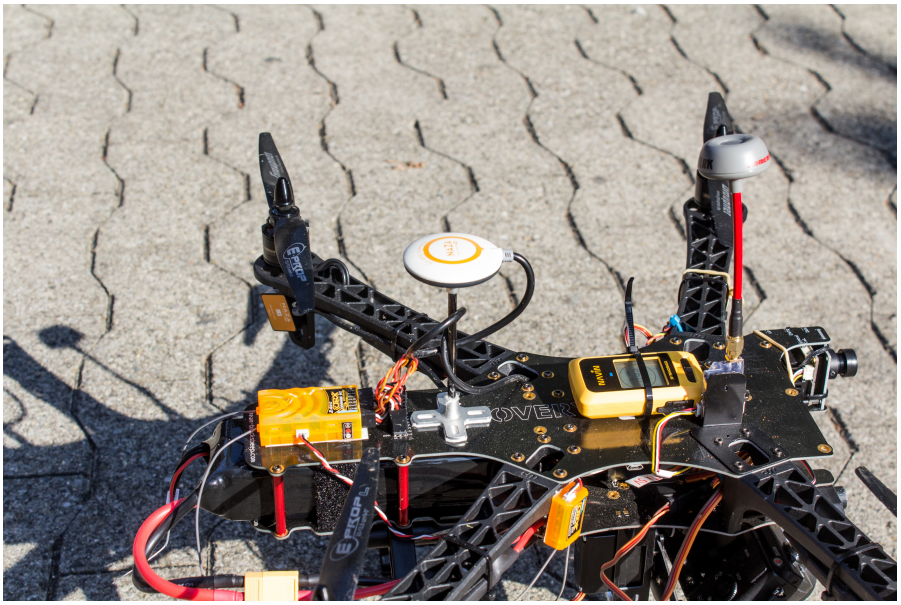


Abbildung 17: Deutlich sichtbar: Der gelbe GPS-Tracker

1 https://www.znex.de/de/minihomer/

Während 10 Minuten flogen wir nun auf einer Höhe zwischen 50m und 120m über das HSR-Gelände, um Bildmaterial von jedem Gebäude zu erhalten.



Abbildung 18: GPS Trace des Quadkopters während der Erfassung.

4.2 GEOCODING DER BILDER

Zurück zuhause sortierten wir die Bilder und löschten unbrauchbares Bildmaterial. Anschliessend exportierten wir den GPS-Trace mithilfe von *GPSBabel*[2]:

```
gpsbabel -i skytraq -f /dev/ttyUSB0 -o gpx -F gps-recording.gpx
```

Anschliessend muss diese Information in die Bild-Metadaten geschrieben werden. Den Aufnahmezeitpunkt der Fotos kann mit dem Speicherzeitpunkt der GPS-Messpunkte korreliert werden um so die entsprechende Position zu finden.

Um dies zu bewerkstelligen, gibt es verschiedene Tools. Wir haben *GPicSync*[3] verwendet, um die Geodaten im IPTC-Format[4] in den Bildern zu speichern. Die verwendeten Einstellungen sind in Abbildung 19 ersichtlich.

Wichtig ist dabei vor allem ein möglicher Offset zwischen Kamera-Uhrzeit und GPS-Uhrzeit. Man kann diese Abweichung relativ einfach berechnen, indem man ein Foto des GPS-Geräts erstellt, auf welchem die genaue Uhrzeit ablesbar ist. Der Offset ist die Differenz zwischen der Uhrzeit auf dem Foto und der Uhrzeit in den Bild-Metadaten.

In *GPicSync* kann der Offset über `Options > Local time correction` konfiguriert werden, indem man die Uhrzeit beider Geräte eintippt.

2 http://www.gpsbabel.org/
3 https://github.com/metadirective/gpicsync
4 https://de.wikipedia.org/wiki/IPTC-IIM-Standard

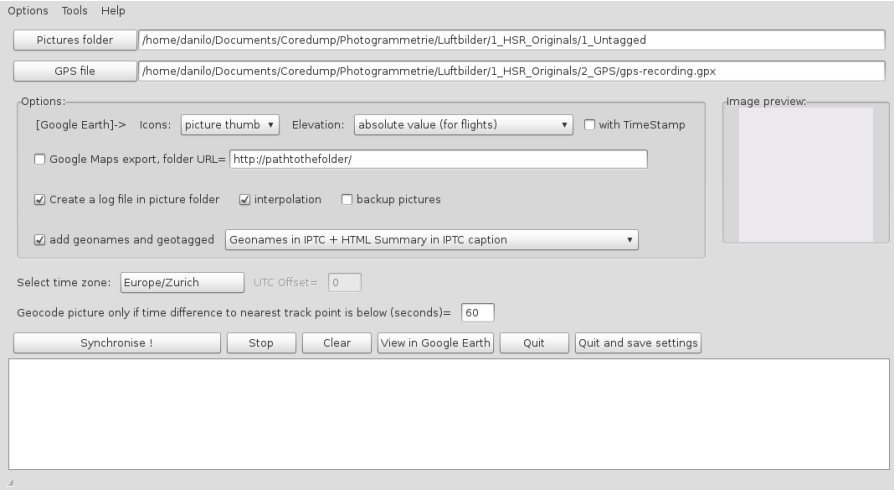


Abbildung 19: Einstellungen GPicSync

4.3 EINFÜHRUNG OPENDRONEMAP

OpenDroneMap (https://github.com/OpenDroneMap/OpenDroneMap) ist ein Projekt, welches im Jahr 2014 von Stephen Mather ins Leben gerufen wurde[8]. Es beinhaltet eine Sammlung von Scripts, um Bilder von Multikoptern, Heissluftballonen, Drachen oder anderen Luftfahrzeugen in diverse Formate zu konvertieren.

Der aktuelle Funktionsumfang beinhaltet als Zielformate Punktwolken, Oberflächenmodelle, Texturierte Oberflächenmodelle und Orthofotos. Zudem kann OpenDroneMap auch Geodaten aus den EXIF- oder IPTC-Daten der Bilder verwenden. Alternativ kann auch eine Datei mit Vollpasspunkten (Englisch: Ground Control Points) bereitgestellt werden, welche dann für die Georeferenzierung verwendet wird.

Um dies zu erreichen, integriert OpenDroneMap viele existierende Open Source Libraries wie CMVS, PMVS, Graculus und weitere. Es gibt also eine deutliche Überschneidung mit der in VisualSFM verwendeten Toolchain aus Abschnitt 3.4.

Anders als VisualSFM ist OpenDroneMap jedoch kein GUI-Programm, sondern wird rein auf der Kommandozeile ausgeführt. Viele Parameter ermöglichen die Anpassung und Optimierung des Prozesses.

Das Bemerkenswerte an OpenDroneMap ist, dass es die erste Open Source Toolchain seiner Art ist. Bisher gab es keinerlei freie Software, die die ganze Photogrammetrie-Pipeline von den Fotos bis zum Orthofoto und Oberflächenmodell abgedeckt haben. Kommerzielle Alternativen dazu sind teure Programme wie Pix4D oder Agisoft PhotoScan.

OpenDroneMap hat eine aktive Community, seit 2014 haben bereits 30 verschiedene Personen Code dazu beigesteuert. Es lohnt sich, die Entwicklung des Projektes weiter zu verfolgen.

4.4 INSTALLATION OPENDRONEMAP

4.4.1 *Lokale Installation*

OpenDroneMap setzt eine Ubuntu 12.04 oder 14.04 Installation voraus. Das Projekt liefert ein Installations-Script mit, welches alle benötigten Abhängigkeiten installiert und dann die mitgelieferten Komponenten kompiliert.

```
git clone https://github.com/OpenDroneMap/OpenDroneMap
cd OpenDroneMap
./install.sh
```

Nun kann das Haupt-Script von einem Bilder-Ordner aus aufgerufen werden.

```
cd /path/to/images
/path/to/opendronemap/run.pl
```

Der Nachteil dieser Methode liegt darin, dass man dem Script vertrauen muss, dass es keine «Unordnung» hinterlässt. Es gibt keinen Uninstaller.

4.4.2 *Vagrant / VirtualBox*

Auf https://github.com/OpenDroneMap/odm_vagrant findet sich eine Anleitung, wie man mithilfe von Vagrant[5] eine VirtualBox[6] VM starten und initialisieren kann, welche dann OpenDroneMap ausführt.

Der Vorteil davon ist die Isolation des Systemes - wenn man die Software nicht mehr benötigt kann man einfach die VM löschen. Zudem läuft die Virtuelle Maschine auch auf Systemen die OpenDroneMap nicht unterstützen, wie Windows oder OS X. Nachteil ist vor Allem der grosse Overhead der Virtualisierung.

4.4.3 *Docker*

Das beste aus zwei Welten bietet Docker[7], eine Software für die Erstellung und den Betrieb von Linux-Containern auf Basis von LXC oder libcontainer. Dabei wird nicht die ganze Hardware virtualisiert, sondern lediglich das Betriebssystem. Der Kernel hingegen wird mit dem Host-System geteilt. Um dies zu erreichen, werden Linux Kernel-Features wie Control Groups und Kernel Namespaces verwendet.

5 https://www.vagrantup.com/
6 https://www.virtualbox.org/
7 https://www.docker.com/

Diese Lösung erlaubt die Isolation einer Applikation, hat aber einen extrem geringen Overhead. Der Container startet innerhalb von Millisekunden, nicht innerhalb von Minuten wie bei VirtualBox.

OpenDroneMap verfügt bisher über kein offizielles Docker-Image, wir haben jedoch das dafür benötigte Dockerfile geschrieben und dem Projekt einen Pull Request gesendet[8]. Inzwischen wurde der Pull Request gemerged, daher kann diese Deployment-Methode nun sehr einfach getestet werden:

```
git clone -b docker https://github.com/OpenDroneMap/OpenDroneMap
cd OpenDroneMap
export IMAGES=/absolute/path/to/your/images
docker build -t opendronemap:latest .
docker run -v $IMAGES:/images opendronemap:latest
```

Docker läuft nativ nur unter Linux. Für Windows und OS X Systeme gibt es jedoch mit Boot2Docker eine einfache Möglichkeit, Docker trotzdem zu nutzen:

- https://docs.docker.com/docker/installation/windows/
- https://docs.docker.com/docker/installation/mac/

4.5 GENERIERUNG ORTHOFOTO UND OBERFLÄCHENMODELL

Wie im letzten Kapitel bereits gezeigt, wird OpenDroneMap mithilfe des `run.pl` Scripts aus dem Bilder-Ordner heraus gestartet.

Zuerst muss man also einen Ordner mit allen Bildern vorbereiten. Wir haben ein geeignetes Subset der zuvor in Abschnitt 4.2 geocodierten Bilder (124 Stück) ausgewählt und in einen separaten Ordner kopiert.

Sehr wichtig ist, dass die Bilder über eine klein geschriebene Dateiendung verfügen. Aufgrund eines Bugs in OpenDroneMap[9] können die Bilder ansonsten aktuell nicht verarbeitet werden. Falls die Bilder aktuell noch über eine gross geschriebene Endung verfügen, können sie mit folgendem Befehl einfach umbenannt werden:

```
for file in *.JPG; do mv "$file" "`basename $file .JPG`.jpg"; done
```

Anschliessend kann OpenDroneMap gestartet werden. Da wir keine Ground Control Points bereitstellen, muss noch die entsprechende Option (`--odm_georeferencing-useGcp false`) mitgegeben werden.

8 https://github.com/OpenDroneMap/OpenDroneMap/pull/143
9 https://github.com/OpenDroneMap/OpenDroneMap/issues/33

```
# Normale Installation
/path/to/opendronemap/run.pl --odm_georeferencing-useGcp false

# Docker
docker run --rm -v $(pwd):/images opendronemap:latest \
       --odm_georeferencing-useGcp false
```

Der Prozess dauerte bei unserem Datensatz knapp 3 Stunden.

Falls stattdessen die Fehlermeldung *«no CCD width or focal length found»* ausgegeben wird, fehlt das Kameraprofil der genutzten Kamera in OpenDroneMap. Das Problem kann jedoch schnell beseitigt werden, siehe Abschnitt C.1.

4.6 RESULTATE

Die Resultate der Verarbeitung befinden sich in einem Unterordner des Bildverzeichnisses mit dem Namen `reconstruction-with-image-size-2400-results`.

4.6.1 *Orthofoto*

Das Orthofoto hat den Namen `odm_orthophoto.png`. In unserem Fall war es 14'236 × 14'143 Pixel (ca. 200 Megapixel) und 120 MiB gross.



Abbildung 20: Resultierendes Orthofoto des HSR-Areals

Wie man sieht, ist das Luftbild orthorektifiziert, d.h. die geometrischen Verzerrungen sind aus den Rohdaten entfernt und die Rotation stimmt mit den effekiven Himmelsrichtungen überein. Dies ist dank den eingebetteten GPS-Metadaten möglich.

4.6.2 *Texturiertes Oberflächenmodell*

OpenDroneMap erzeugt nicht nur ein 2D-Luftbild, sondern auch ein orthorektifiziertes und texturiertes 2.5D-Oberflächenmodell. Dieses befindet sich im Unterordner `odm_texturing`. Das Modell (`odm_texturing/odm_textured_model_geo.obj`) kann beispielsweise mit MeshLab geöffnet werden.

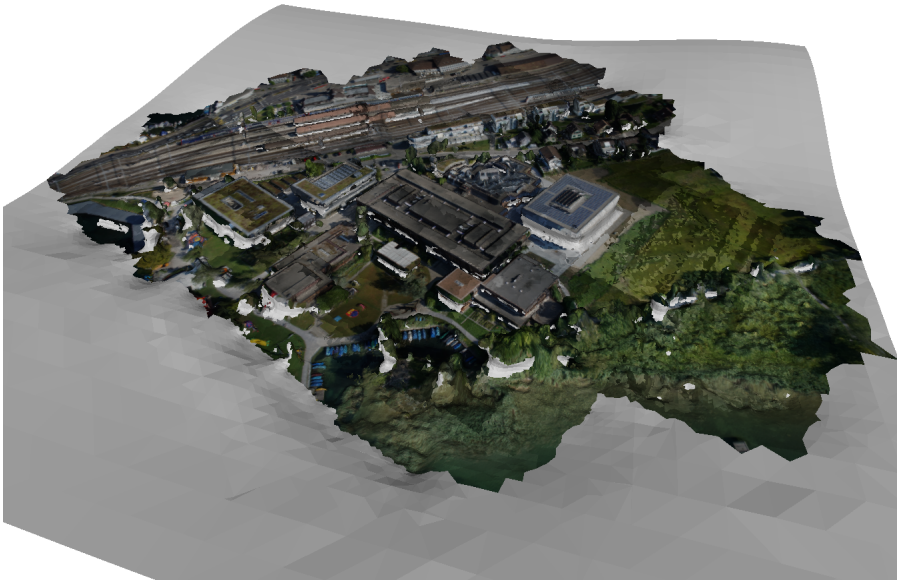


Abbildung 21: Resultierendes Oberflächemodell des HSR-Areals

Das untexturierte Mesh kann online unter `https://sketchfab.com/models/3dd7e0e053054e06b526d2c478253cef` betrachtet werden. (Die texturierte Variante ist zu gross für einen Gratis-Account auf Sketchfab.)

4.6.3 *Beurteilung der Resultate*

- Dafür, dass wir OpenDroneMap in der Standardkonfiguration ohne weitere Anpassungen über die Bilder haben laufen lassen, sehen die Resultate sehr beeindruckend aus. Was jedoch sofort auffällt, ist die unregelmässige Helligkeit des Bildes. Hier fehlt noch etwas Arbeit bei der Normalisierung und Überblendung

der Farben und Helligkeit der einzelnen Bilder. Daran wird jedoch bereits gearbeitet.[10]

- OpenDroneMap berechnet bisher noch keine Ground Sampling Distance (GSD)[11]. Die GSD ist die Distanz zwischen zwei nebeneinanderliegenden Pixel, gemessen auf dem Boden. Das Resultat ist abhängig von der Flughöhe, der Sensorbreite und der Brennweite der Kamera.
  Bei der Software-Evaluation (siehe Kapitel 7) haben wir auch ein Orthofoto mit Pix4D generiert; das Resultat wies eine horizontale Auflösung von 20'772 Pixel auf. Bei OpenDroneMap waren es 10'112 Pixel, also knapp die Hälfte. Da Pix4D im Schluss-Report eine GSD von 2.21cm auswies, gehen wir bei OpenDroneMap von einer GSD von ungefähr 5cm aus.

- Das Orthofoto ist nicht georeferenziert, sondern lediglich orthorektifiziert: Die Himmelsrichtungen stimmen mit der Ausrichtung des Bildes überein. Für eine korrekte Georeferenzierung müsste man OpenDroneMap eine Liste von Ground Control Points übergeben (siehe Abschnitt 4.5). In einer zukünftigen Version von OpenDroneMap wird Georeferenzierung über GPS-Daten noch nachgerüstet werden[12]. Alternativ kann man das Bild mit einem Tool wie QGIS[13] georeferenzieren.

4.6.4 *Learnings*

Rückblickend gäbe es ein paar Dinge, die wir bei einem zweiten Versuch verbessern würden:

- Das Orthofoto weist immer noch Lücken auf. Bei einem zweiten Versuch würden wir mehr Bildmaterial sammeln.

- Wir haben rein senkrechte Bilder erstellt. Das reicht zwar für das Orthofoto aus, im texturierten Oberflächenmodell sieht man jedoch viele weisse, unförmige Wände. Dies würde sich durch eine 45°-Erfassung des Geländes wohl stark verbessern.

4.6.5 *Rohdaten*

Alle Rohdaten können vom Coredump Fileserver heruntergeladen werden. Weitere Infos im Anhang A.

10 https://github.com/OpenDroneMap/OpenDroneMap/issues/81
11 https://github.com/OpenDroneMap/OpenDroneMap/issues/108
12 https://github.com/OpenDroneMap/OpenDroneMap/issues/144
13 http://qgis.org/

4.7 WEITERE RESSOURCEN

Hier listen wir ein paar weitere Ressourcen auf, die wir während unserer Arbeit gefunden haben.

4.7.1 *Mapknitter*

Auf http://mapknitter.org/ existiert ein Projekt zur einfachen Erstellung von Luftbildern (aerial maps). Der Workflow ist jedoch manuell, d.h. die Luftbilder werden auf eine Karte gezogen und dort «von Hand» ausgerichtet. Dadurch ist der Prozess etwas aufändiger und liefert andere Resultate als OpenDroneMap.

4.7.2 *OpenAerialMaps*

OpenAerialMaps (http://openaerialmap.org/) ist sich ein Projekt, welches offen lizenzierte Luftbilder sammelt. Der Fokus liegt dabei auf Bildern von unbemannten Luftfahrzeugen, wie unserem Quadrokopter. Es ist quasi das OpenStreetMap von der Orthofotografie.

Teil III

SOFTWARE-EVALUATION

5 EINLEITUNG

In diesem Teil der Dokumentation vergleichen wir verschiedene Photogrammetrie-Lösungen. Berücksichtigt werden folgende Produkte:

Name	**Lizenz**	**Preis**
VisualSFM	Teilweise OSS	Gratis
OpenDroneMap	Freie Software	Gratis
Pix4Dmapper Pro	Proprietär	7900 CHF
PhotoScan Pro	Proprietär	3499 USD

Abbildung 22: Evaluierte Photogrammetrie-Software: Lizenz / Kosten

Die Produkte bieten folgenden Funktionsumfang:

Name	**3D Rekonstruktion**	**Orthofoto**	**DSM**
VisualSFM	Ja	Nein	Jein
OpenDroneMap	Nein	Ja	Ja
Pix4Dmapper Pro	Ja	Ja	Ja
PhotoScan Pro	Ja	Ja	Ja

Abbildung 23: Evaluierte Photogrammetrie-Software: Funktionsumfang

Für die Evaluation werden zwei Datasets verwendet: Ein Dataset von einem Stroh-Hasen in Jona zum Testen der 3D-Rekonstruktion und ein Dataset mit Senkrecht-Fotos des HSR-Geländes zum Testen der Orthofoto- und DSM-Generierung.

Verglichen werden dabei die Dauer der Rekonstruktion, die Anzahl der generierten Punkte in der Punktwolke, die Anzahl der Vertizes im generierten Mesh und die Resultate selbst.

Alle Tests wurden auf einem System mit einem Intel Core i7-4770 3.4 GHz Prozessor und 32 GiB RAM durchgeführt. Für die Rekonstruktion wurden in jeder Software die Standardeinstellungen verwendet.

6 3D REKONSTRUKTION

6.1 WAHL DES TEST-OBJEKTS

Zum Vergleichen der 3D-Rekonstruktions-Fähigkeiten verschiedener Softwarelösungen haben wir uns ein geeignetes Objekt gesucht: Freistehend, unregelmässig geformt und nicht ganz trivial rekonstruierbar. In der Nähe des Bächlihofes in Jona wurden wir fündig: Dort stand ein grosser, für unsere Zwecke gut geeigneter Stroh-Hase.

Wie auch im Abschnitt 4.1 beschrieben, erfassten wir den Hasen mit einem TBS Discovery Pro Quadrokopter und einer Sony Alpha 5100 Kamera. Für dieses Dataset speicherten wir jedoch keine GPS-Daten, diese wären auf dieser kleinen Fläche wohl nicht genügend genau.

Das resultierende Dataset enthält 315 Bilder, gesamthaft 3.1 GiB.



Abbildung 24: Erfassung Bildmaterial Strohhase

6.2 RESULTATE

6.2.1 *Rekonstruktionsdauer*

Folgende Tabelle vergleicht die Rekonstruktionsdauer für die Sparse Cloud, die Dense Cloud und – wo von der Software unterstützt – das Mesh.

Software	**Sparse Cloud**	**Dense Cloud**	**Mesh**
VisualSFM	1h 13m	1h 49m	n/a
Pix4Dmapper Pro	1h 40m	3h 46m	13m
PhotoScan Pro	3h 50m	8h 11m	8m

6.2.2 *Komplexität der Resultate*

Als Vergleichsmass für die Komplexität der Resultate haben wir die Anzahl der Punkte in der verdichteten Punktwolke sowie die Anzahl der Vertizes im Mesh verwendet.

Dies ist natürlich nur bedingt aussagekräftig, da die Anzahl der Punkte bzw. Vertizes von den Konversions-Parametern abhängig ist und auch nicht zwingend etwas über die Qualität des Modells aussagt.

Software	**Punkte in Punktwolke**	**Vertizes in Mesh**
VisualSFM	4'998'104	n/a
Pix4Dmapper Pro	27'921'022	494'359
PhotoScan Pro	11'823'773	397'244

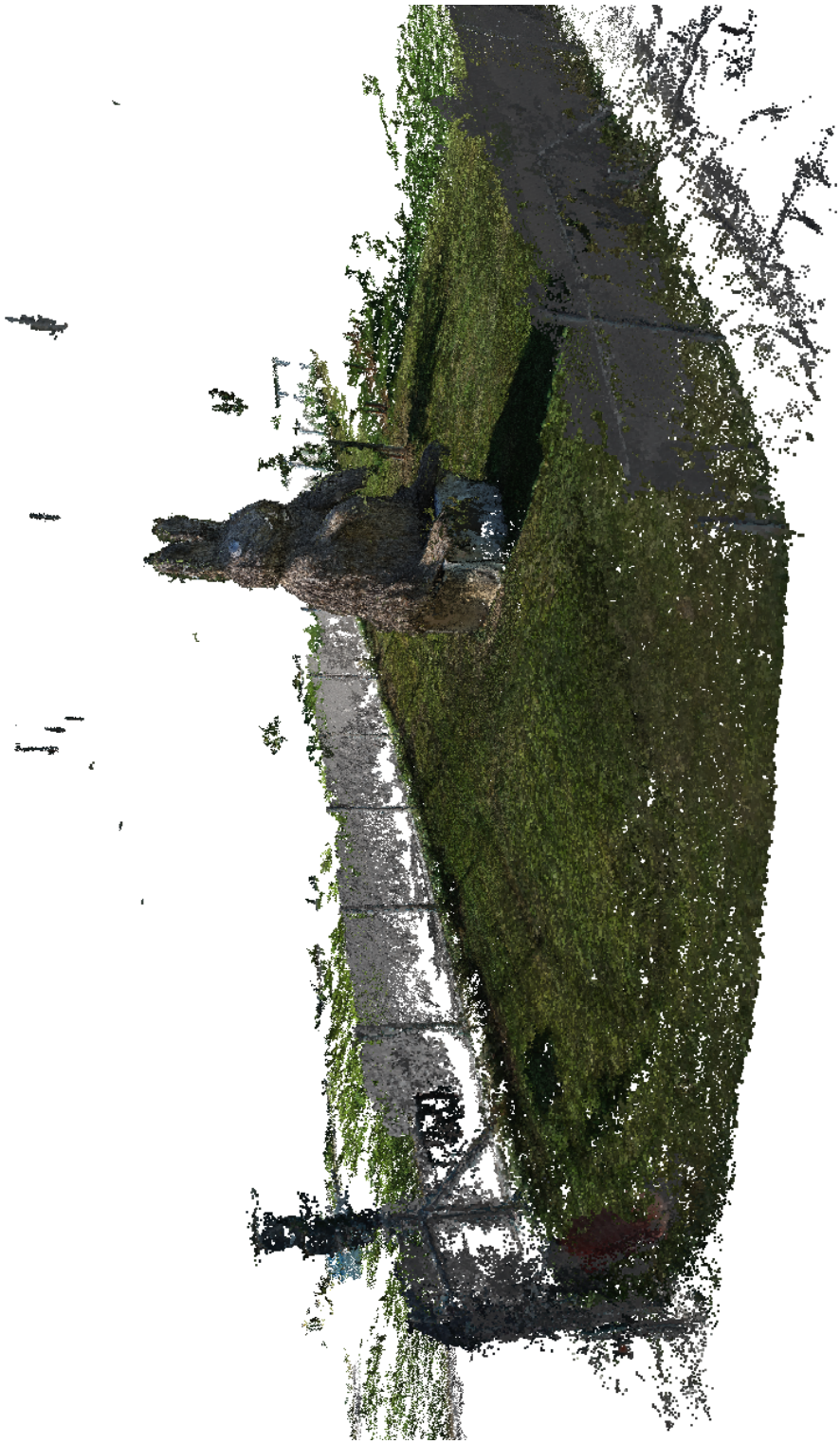


Abbildung 25: Resultat: Punktwolke Hase mit VSFM



Abbildung 26: Resultat: Punktwolke Hase mit Pix4DMapper Pro



Abbildung 27: Resultat: Punktwolke Hase mit PhotoScan Pro

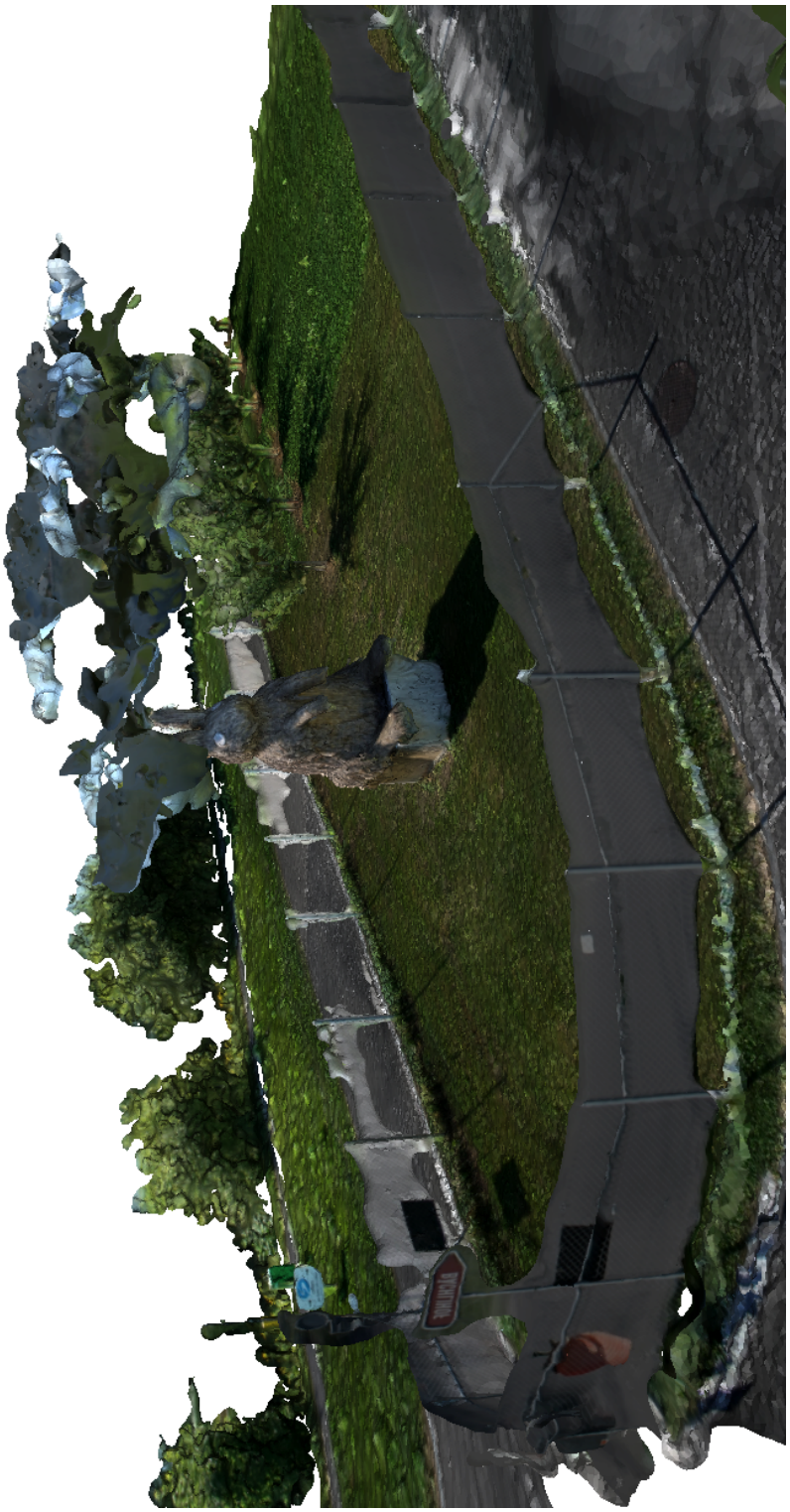


Abbildung 28: Resultat: Mesh Hase mit Pix4DMapper Pro

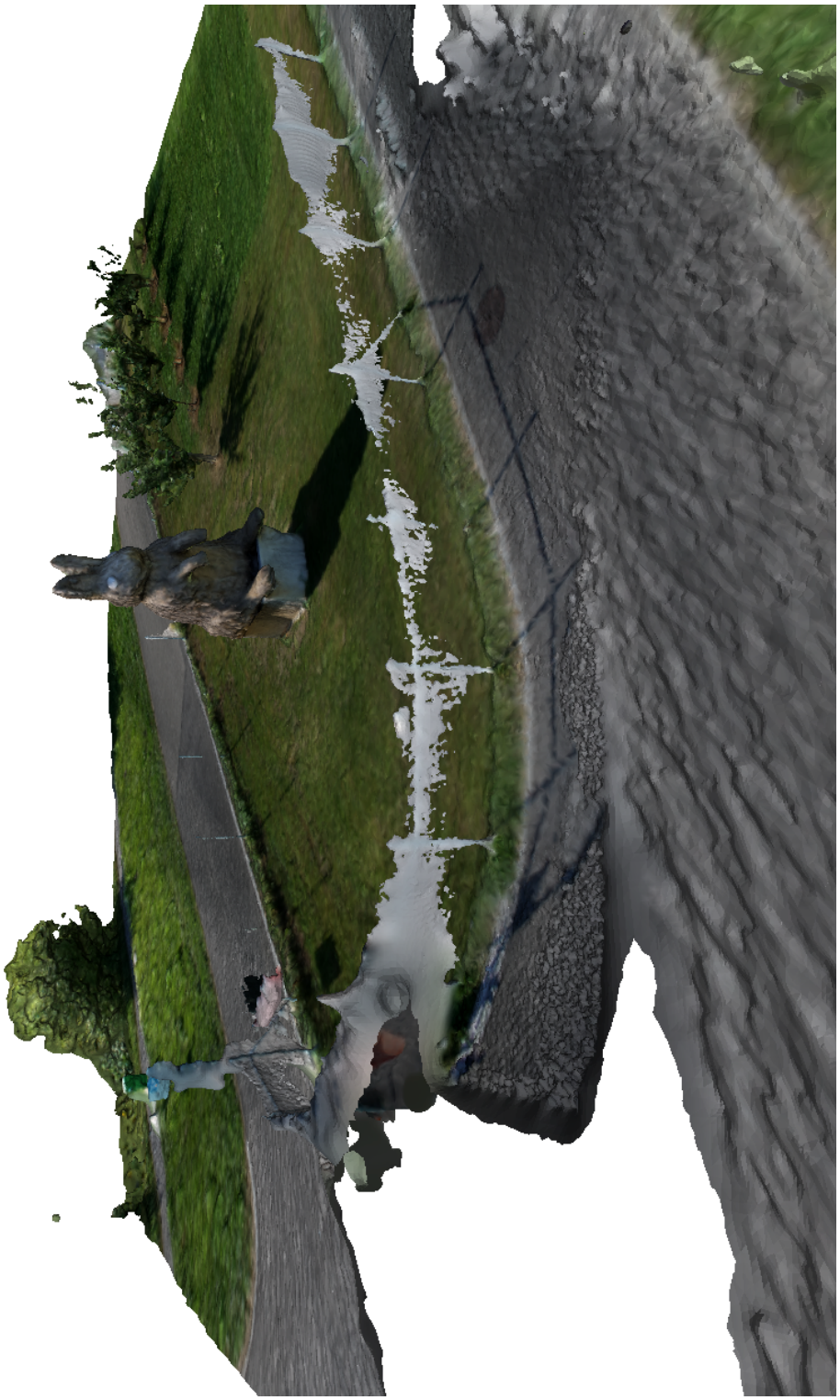


Abbildung 29: Resultat: Mesh Hase mit PhotoScan Pro

7

ERSTELLUNG ORTHOFOTO UND OBERFLÄCHENMODELL

7.1 WAHL DES TEST-OBJEKTS

Zum Testen der Orthofoto- und Oberflächenmodell-Erstellung verschiedener Softwarelösungen haben wir uns für das HSR Gelände entschieden. Am Tag der Luftaufnahmen waren zudem Aufbauarbeiten für das Seenachtsfest in Gange, daher sieht man auch einige Hüpfburgen und Zelte auf den Luftbildern.

Die Erfassung der Bilder wurde bereits im Abschnitt 4.1 beschrieben. Alle Bilder wurden mit GNSS-Koordinaten versehen.

Das resultierende Dataset enthält 124 Bilder, gesamthaft 1.2 GiB.



Abbildung 30: Erfassung Bildmaterial HSR

7.2 RESULTATE

7.2.1 *Rekonstruktionsdauer*

Folgende Tabelle vergleicht die Rekonstruktionsdauer für das Oberflächenmodell und für das Orthofoto.

Software	**DSM + Orthofoto**
OpenDroneMap	2h 54min
Pix4Dmapper Pro	11h 16min
PhotoScan Pro	4h 56min

7.2.2 *Komplexität der Resultate*

Als Vergleichsmass für die Komplexität der Resultate haben wir die Anzahl der Pixel im Orthofoto sowie die Anzahl der Vertizes im Oberflächenmodell verwendet.

Dies ist natürlich nur bedingt aussagekräftig, da die Anzahl der Pixel bzw. Vertizes von den Konversions-Parametern abhängig ist und auch nicht zwingend etwas über die Qualität des Bildes bzw. Modells aussagt.

Software	**Pixel in Orthofoto**	**Vertizes in DSM**
OpenDroneMap	10'112 x 9'066	100'050
Pix4Dmapper Pro	20'772 x 15'988	498'202
PhotoScan Pro	23'254 x 19'496	685'956

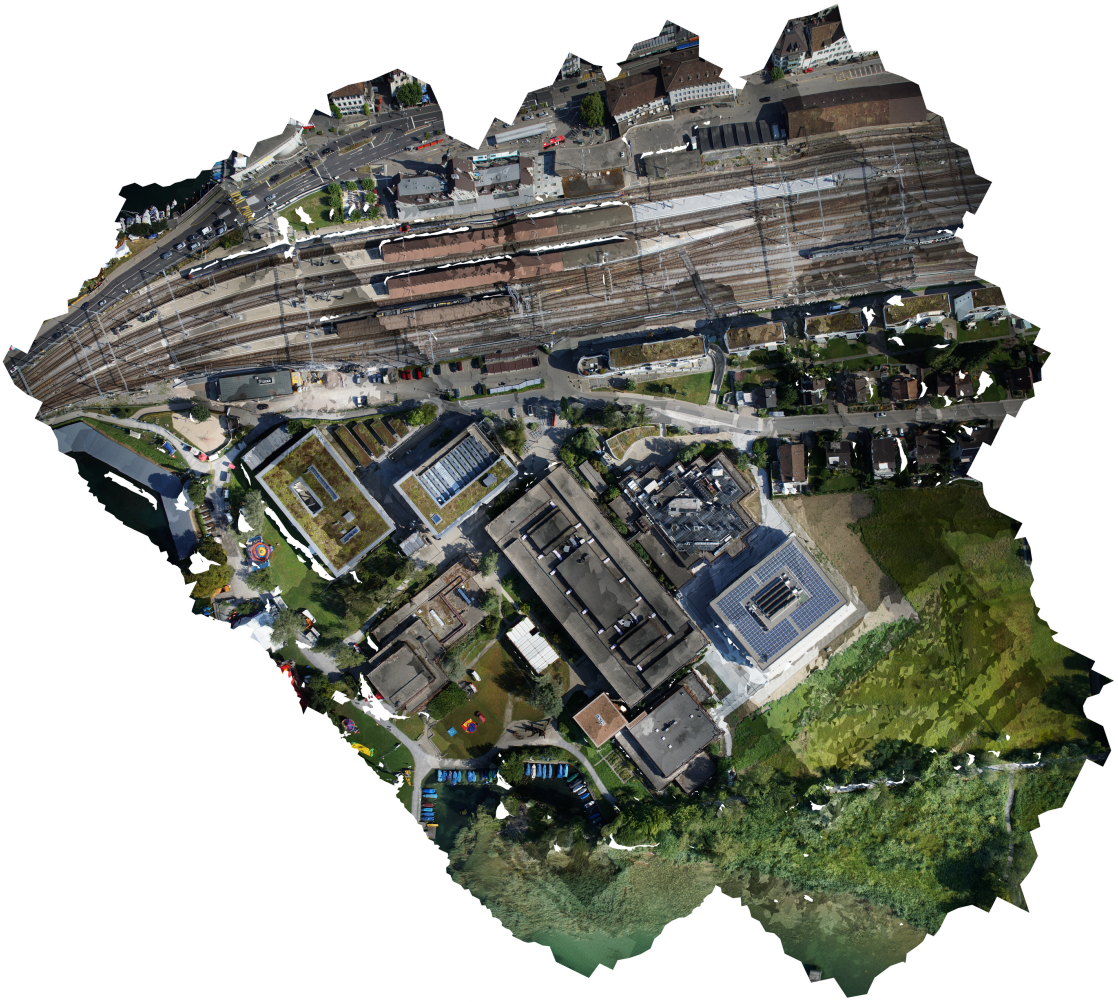


Abbildung 31: Resultat: Orthofoto HSR mit OpenDroneMap

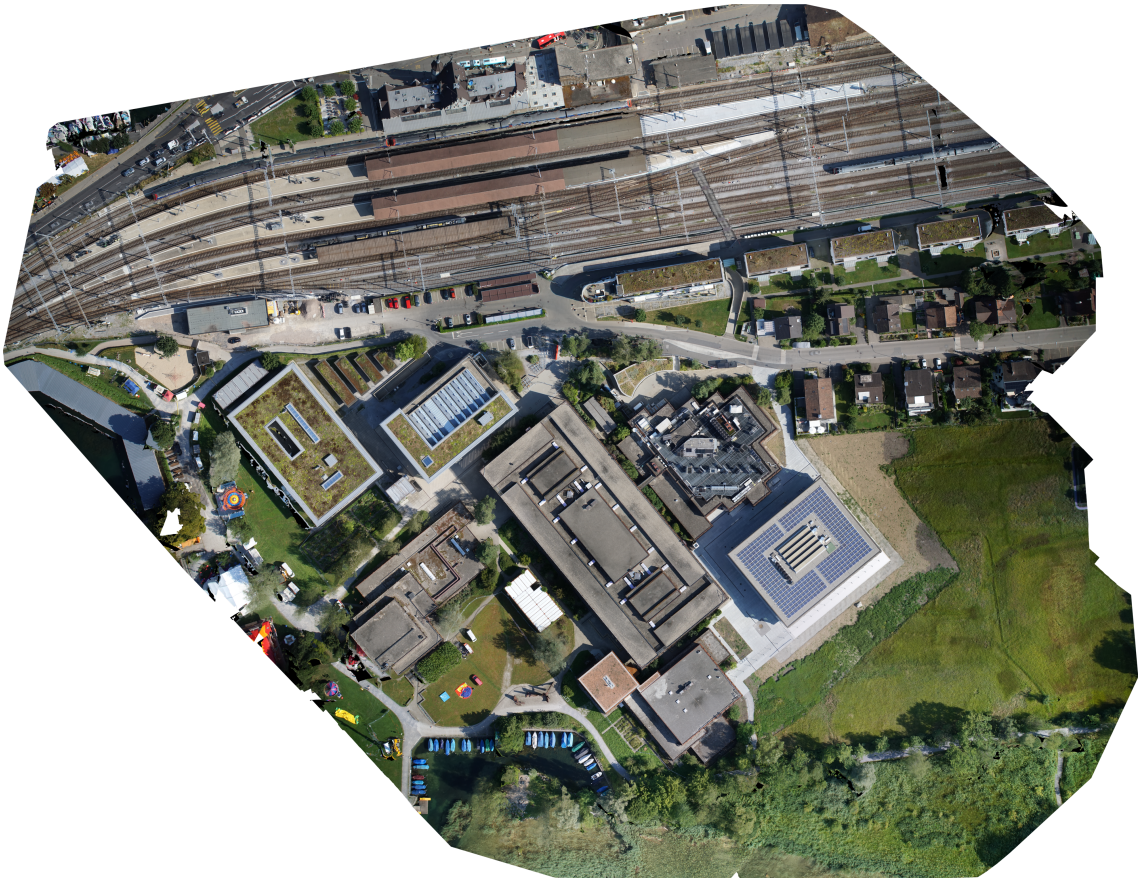


Abbildung 32: Resultat: Orthofoto HSR mit Pix4Dmapper Pro

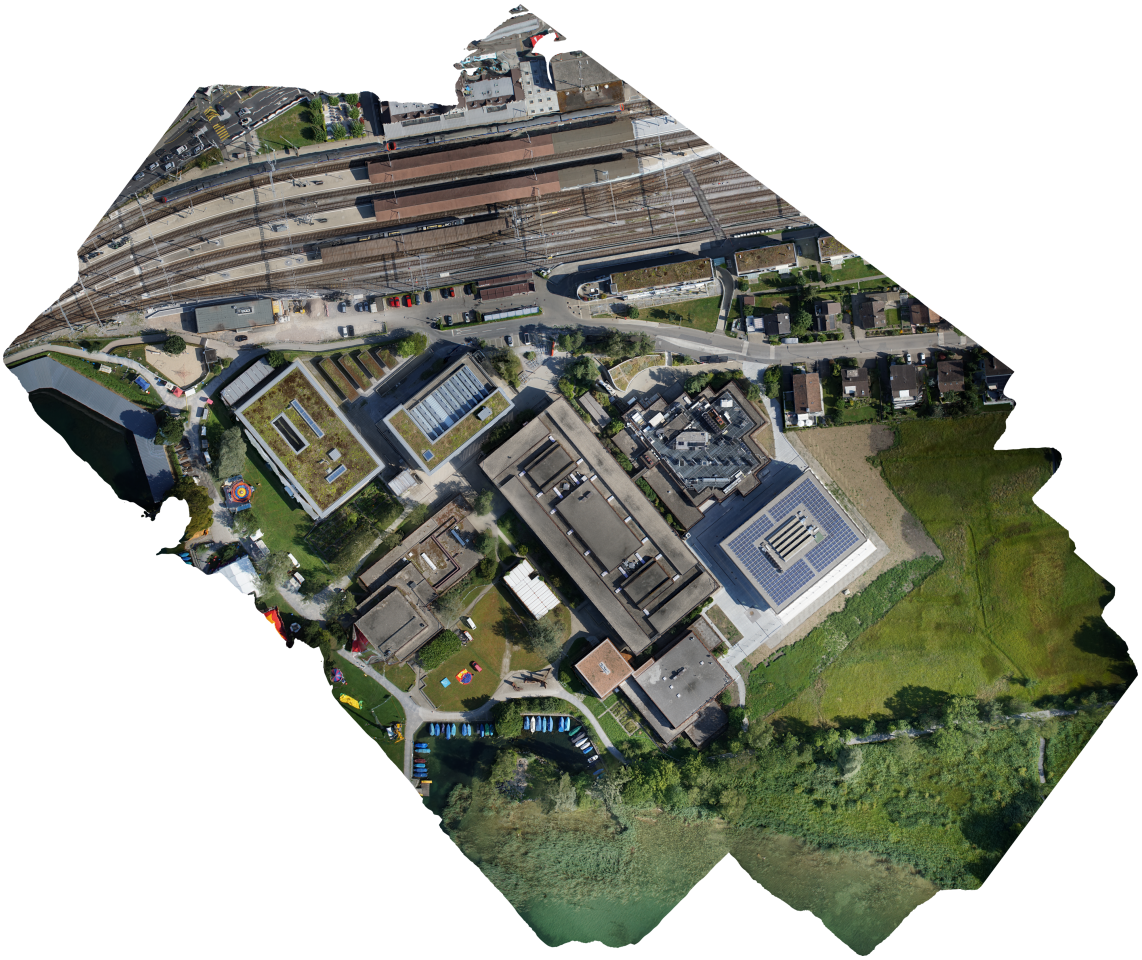


Abbildung 33: Resultat: Orthofoto HSR mit PhotoScan Pro

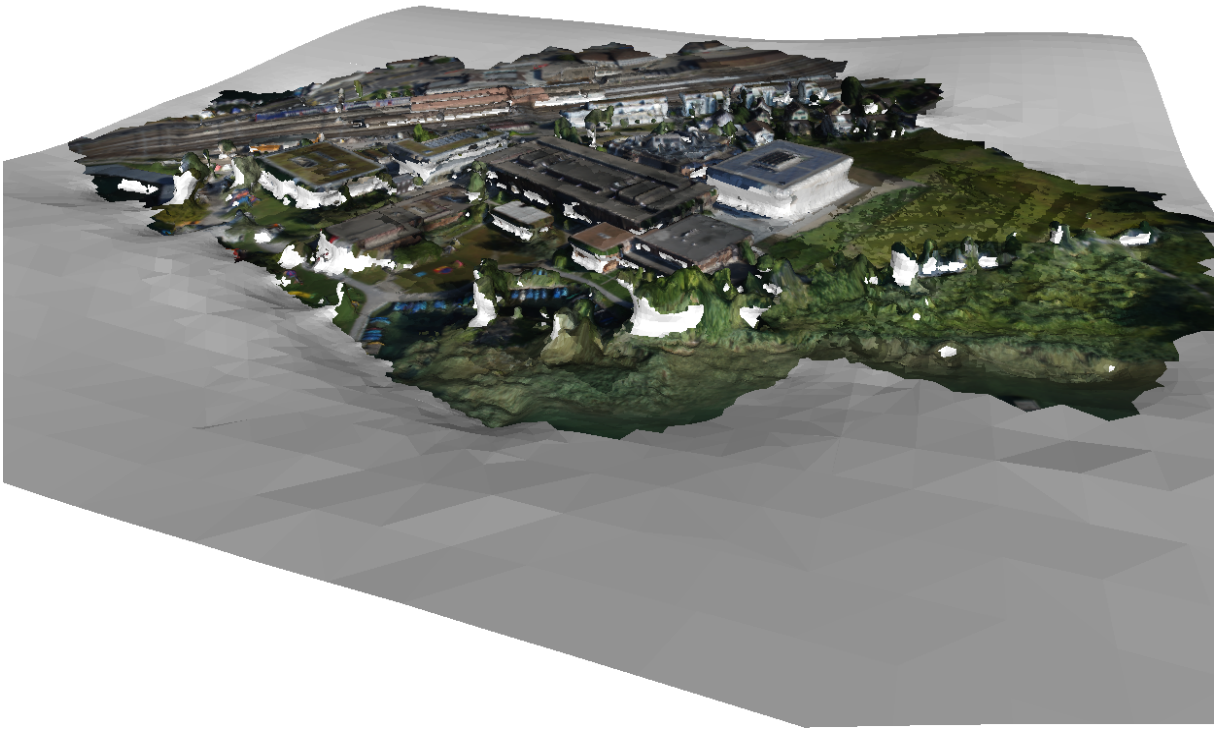


Abbildung 34: Resultat: Oberflächenmodell HSR mit OpenDroneMap



Abbildung 35: Resultat: Oberflächenmodell HSR mit Pix4Dmapper Pro



Abbildung 36: Resultat: Oberflächenmodell HSR mit PhotoScan Pro

Teil IV

APPENDIX

APPENDIX A – FILESERVER / DOWNLOADS

Unter folgender URL sind alle Rohdaten, Projektverzeichnisse und Resultate verfügbar:

`https://photogrammetry.coredump.ch/`

Neben einer direkt durchsuchbaren HTTP-Verzeichnisstruktur bieten wir auf `https://photogrammetry.coredump.ch/` auch einen Torrent an. Wir sind froh, wenn dieser gegenüber dem HTTP-Download bevorzugt wird, um die Traffic-Last auf verschiedene Seeder zu verteilen.

B

APPENDIX B – INSTALLATION VISUALSFM

Die folgende Anleitung beschreibt die Installation von VisualSFM[1] (nachfolgend VSFM genannt) auf einem Computer mit Ubuntu[2] 14.04 Betriebssystem und einer CUDA-fähigen nVidia-Grafikkarte. CUDA-Unterstützung ist optional, aber empfohlen.

B.1 EINLEITUNG

VSFM ist eine GUI-Applikation für 3D-Rekonstruktion mittels «Structure from Motion» (SFM) Technik. Das Projekt integriert viele Einzelkomponenten, die meist von Universitäten entwickelt wurden. Mehr dazu findet sich im Abschnitt 3.4.

B.1.1 *Abhängigkeiten*

Für den VSFM 3D-Reconstruction Prozess werden folgende Bibliotheken benötigt:

- SiftGPU[3]: Eine SIFT[7]-Implementation für GPUs.
- PMVS (Patch-based Multi-view Stereo)[4]: Erstellung einer Point-Cloud aus Kamerabildern. Das Resultat enthält gemeinsame Features als Punkte mit an der Kamera ausgerichteten Normalen.
- CMVS (Clustering Views for Multi-view Stereo)[5]: Ergänzt PMVS um Clustering für schnellere Resultate.
- PBA[6]: Bündelblockausgleichung (Englisch: Bundle Adjustment) auf mehreren CPUs.

Da für die meisten dieser Komponenten nicht als Ubuntu-Pakete verfügbar sind, müssen sie manuell kompiliert werden.

1 http://ccwu.me/vsfm/
2 http://www.ubuntu.com
3 http://www.cs.unc.edu/~ccwu/siftgpu/
4 http://www.di.ens.fr/pmvs/
5 http://www.di.ens.fr/cmvs/
6 http://grail.cs.washington.edu/projects/mcba/

B.1.2 *Alternativen*

VSFM arbeitet mit vielen alternativen SFM-Programmen zusammen oder kann Teile davon selbst benutzen. Sie sind jedoch nicht direkt für den Betrieb von VSFM nötig und wurden nicht näher betrachtet:

- CMP-MVS[7]
- MVE[8]
- SURE[9]
- MeshRecon[10]

B.2 INSTALLATION

Das folgenden Installationsscript bietet eine Hilfe um VSFM auf einem Ubuntu 14.04 System zu installieren.

Im Script werden alle Abhängigkeiten automatisch aus dem Internet heruntergeladen. Sie sind jedoch ebenfalls auf unserem Fileserver (Anhang A) vorhanden, für den Fall dass die Ressourcen nicht mehr online verfügbar sind. Es wird empfohlen nicht das ganze Script auf einmal auszuführen, sondern es Schritt für Schritt durchzugehen. Möglicherweise haben sich Systemvoraussetzungen in der Zwischenzeit geändert (z.B. nVidia-Treiber).

7 http://ptak.felk.cvut.cz/sfmservice/websfm.pl?menu=cmpmvs
8 http://www.gris.informatik.tu-darmstadt.de/projects/multiview-environment/
9 http://www.ifp.uni-stuttgart.de/publications/software/sure/index.en.html
10 http://www-scf.usc.edu/~zkang/software.html

```
# Whole 2D to 3D reconstruction toolchain setup for Ubuntu 14.04 64bit.
# Make sure to enable the multiverse repository!
# Sources:
# - http://ccwu.me/vsfm/install.html#linux
# - http://www.10flow.com/2012/08/15/
# - http://www.di.ens.fr/cmvs/

# update the system
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install build-essential

# install a proprietary nvidia driver
sudo apt-get install nvidia-331-updates

# prepare dependencies of vsfm, siftgpu, mulicore_bundle_adjustment,
#                         pmvs-2, cmvs, graclus
sudo apt-get install libgl-dev libglu-dev libx11-dev \
                     libgtk2.0-dev libglew-dev glew-utils libdevil-dev \
                     libboost-all-dev libatlas-cpp-0.6-dev libatlas-dev \
                     imagemagick libatlas3gf-base libcminpack-dev \
                     libgfortran3 libmetis-edf-dev libparmetis-dev \
                     freeglut3-dev libgsl0-dev
sudo apt-get install unzip
sudo apt-get install nvidia-cuda-toolkit

# compile vsfm (framework for 3d reconstruction from motion)
wget http://ccwu.me/vsfm/download/VisualSFM_linux_64bit.zip
unzip VisualSFM_linux_64bit.zip
cd vsfm
make
cd ..

# compile SiftGPU (feature detection with GPU)
wget -O SiftGPU.zip http://wwwx.cs.unc.edu/~ccwu/cgi-bin/siftgpu.cgi
unzip SiftGPU.zip
cd SiftGPU
sudo ln -s /usr/lib/nvidia-cuda-toolkit /usr/local/cuda
make
cd ..
cp SiftGPU/bin/libsiftgpu.so vsfm/bin/

# pba (fast bundle adjustments library)
wget http://grail.cs.washington.edu/projects/mcba/pba_v1.0.5.zip
unzip pba_v1.0.5.zip
cd pba
sed -i.bak 's|CUDA_INSTALL_PATH = /usr/local/cuda|CUDA_INSTALL_PATH \
           = /usr/lib/nvidia-cuda-toolkit|g' ./makefile
make
cd ..
cp pba/bin/libpba.so vsfm/bin/

# pmvs-2 (builds a point cloud of features seen in multiple views)
sudo apt-get install liblapack-dev
wget http://www.di.ens.fr/pmvs/pmvs-2-fix0.tar.gz
tar xf pmvs-2-fix0.tar.gz
cd pmvs-2/program/main/
make clean

wget http://www.10flow.com/wp-content/uploads/2012/08/mylapack.o.tar.gz
tar xf mylapack.o.tar.gz
make depend
make
cd ../../../
```

```
# graclus (fast graph clustering library)
wget http://www.cs.utexas.edu/users/dml/Software/graclus1.2.tar.gz
tar xf graclus1.2.tar.gz
cd graclus1.2/
sed -i.bak 's|COPTIONS = -DNUMBITS=32|COPTIONS \
           = -DNUMBITS=64|g' Makefile.in
make
cd ..

# cmvs (cluster pmvs output)
wget http://www.di.ens.fr/cmvs/cmvs-fix2.tar.gz
tar xf cmvs-fix2.tar.gz
cp pmvs-2/program/main/mylapack.o cmvs/program/main/
sed -i.bak '1s/^/#include <vector>\n#include <numeric>\n/' \
           cmvs/program/base/cmvs/bundle.cc
sed -i.bak '1s/^/#include <stdlib.h>\n/' cmvs/program/main/genOption.cc
sed -i.bak 's|YOUR_INCLUDE_METIS_PATH =|YOUR_INCLUDE_METIS_PATH \
           = -I'$(pwd)'/graclus1.2/metisLib|g' \
           cmvs/program/main/Makefile
sed -i.bak 's|YOUR_LDLIB_PATH =|YOUR_LDLIB_PATH \
           = -L'$(pwd)'/graclus1.2|g' cmvs/program/main/Makefile
sed -i.bak 's/^Your\ /# Your /g' cmvs/program/main/Makefile
cd cmvs/program/main
make

cp cmvs ../../../vsfm/bin
cp pmvs2 ../../../vsfm/bin
cp genOption ../../../vsfm/bin

# set export paths
echo "" >> ~/.bashrc
echo export PATH=\$PATH:$(pwd)/vsfm/bin >> ~/.bashrc
echo export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:$(pwd)/vsfm/bin \
            >> ~/.bashrc
```

C

APPENDIX C – OPENDRONEMAP TIPPS

C.1 KAMERAPROFIL ERSTELLEN

Damit OpenDroneMap die übergebenen Bilder richtig verarbeiten kann, benötigt die Software Informationen über die Abmessungen des Kamera-Sensors. Diese Informationen sind für viele Kameramodelle bereits in OpenDroneMap registriert.

Falls ein Kameramodell noch fehlt, wird die folgende Fehlermeldungen erzeugt:

```
no CCD width or focal length found for DSC05399.JPG \
  - camera: "SONY ILCE-5100"
found no usable images - quitting
```

Wie aus der Meldung ersichtlich, wurde das Bild mit einer SONY ILCE-5100 Kamera erstellt, welche OpenDroneMap noch nicht bekannt ist. Eine kurze Google-Suche ergibt, dass die SONY ILCE-5100 identisch ist mit der SONY A5100.

Die Sensorgrösse einer Kamera findet man relativ einfach auf der Website `dpreview.com`. Die SONY A5100 ist vertreten: `http://www.dpreview.com/products/sony/slrs/sony_a5100`

In der «Quick Specs» Liste wird unter «Sensor size» die Sensorgrösse aufgelistet:



Abbildung 37: Sensorgrösse SONY A5100

Den grösseren der zwei Werte muss man nun im OpenDroneMap Sourcecode in die Datei `ccd_defs.json` eintragen:

```
{
  // ...
  "SONY ILCE-5100": 23.5,
  // ...
}
```

Nun OpenDroneMap erneut starten und die Fehlermeldung sollte verschwinden.

D

APPENDIX D – GRUNDRISSE SCHLOSS RAPPERSWIL

Die folgenden Grundrisse haben wir freundlicherweise von der Ortsgemeinde Rapperswil erhalten.

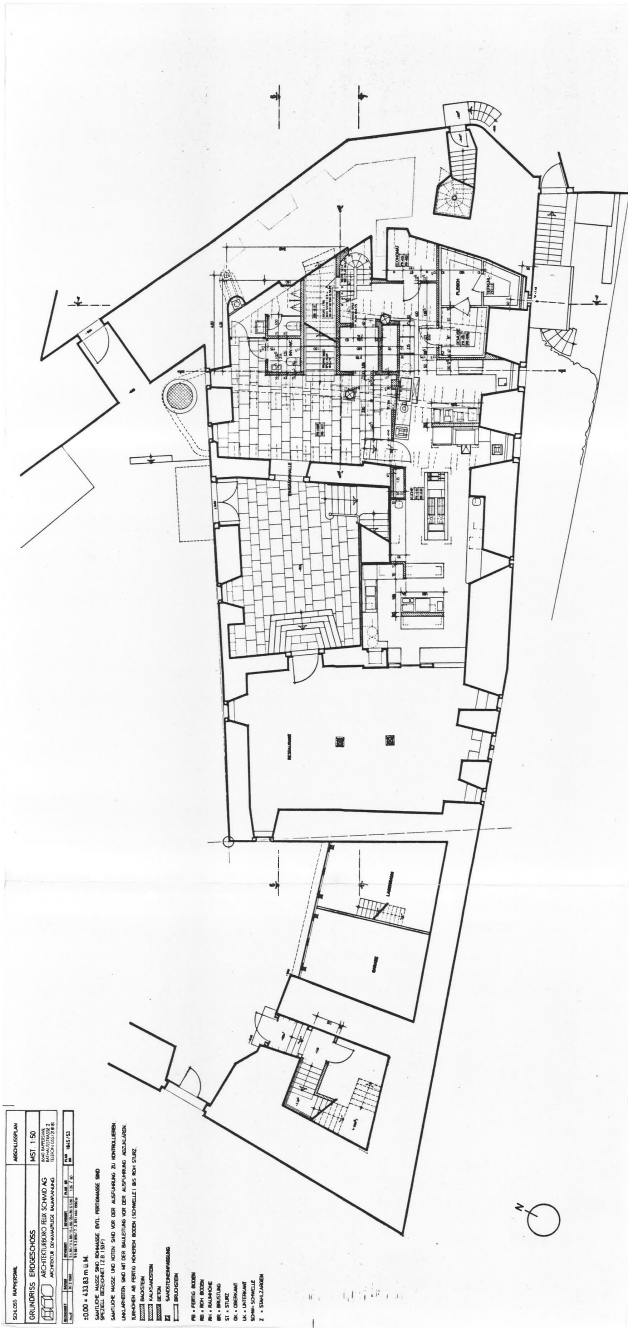


Abbildung 38: Schloss Rapperswil: Grundriss EG

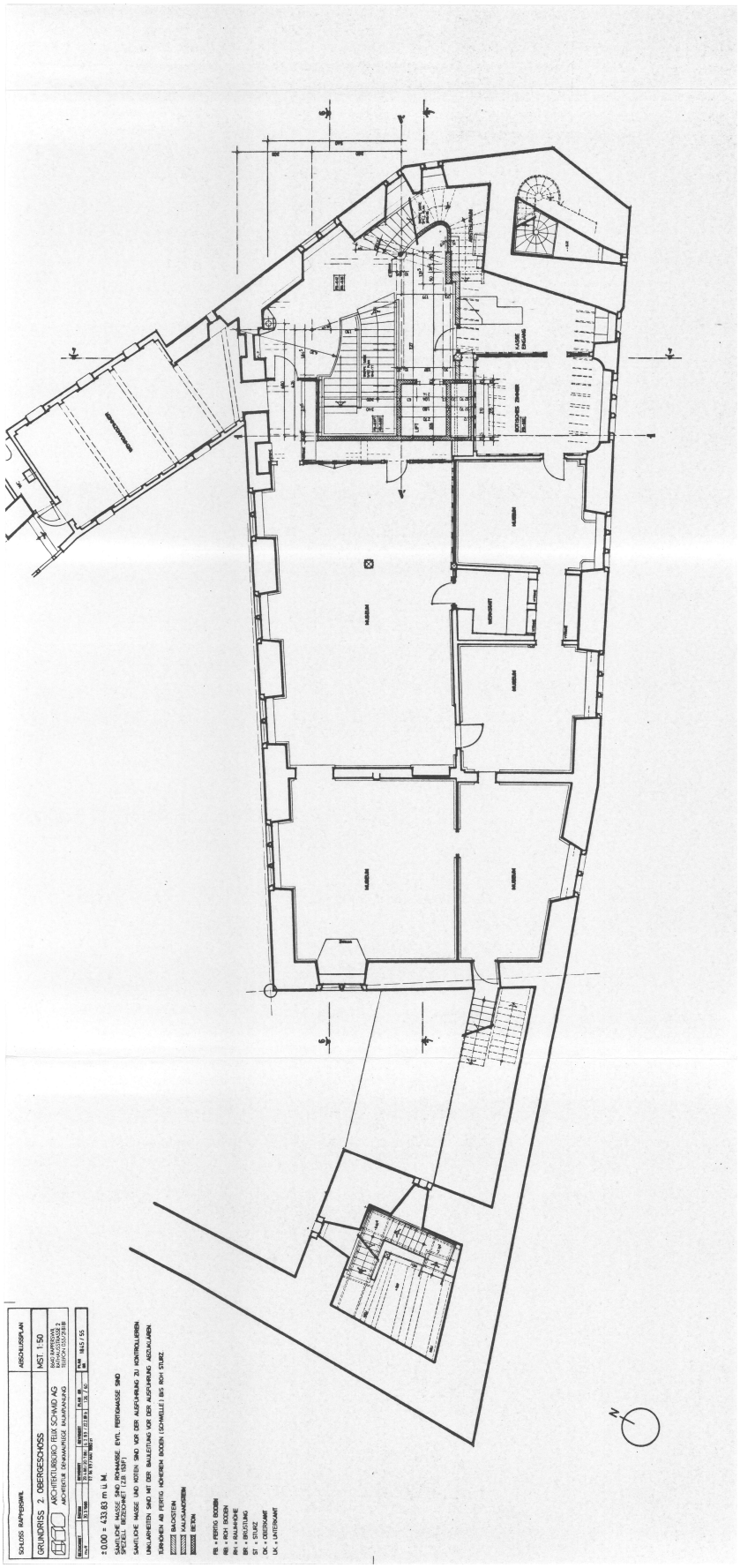


Abbildung 39: Schloss Rapperswil: Grundriss 2. OG

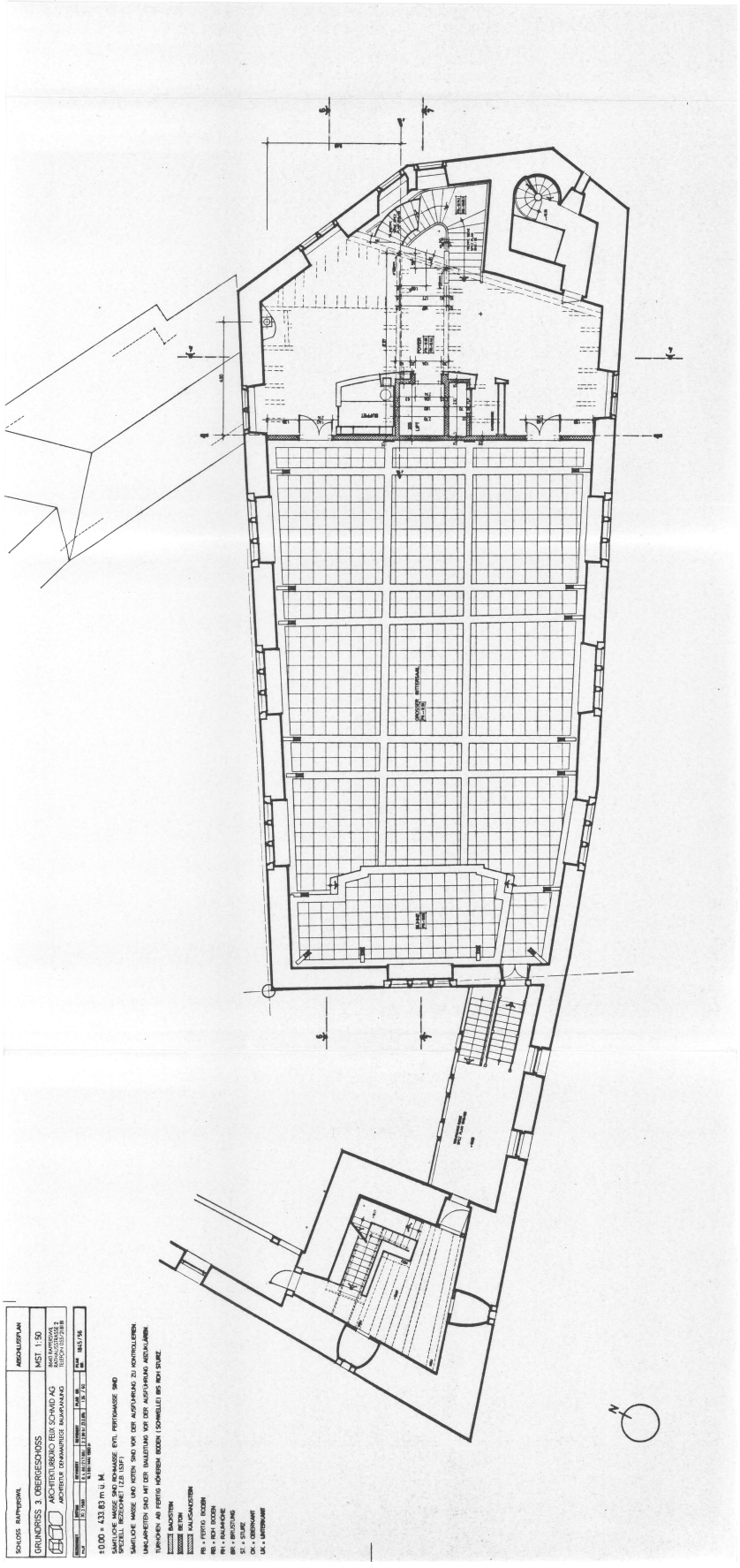


Abbildung 40: Schloss Rapperswil: Grundriss 3. OG

APPENDIX E – SOFTWARELISTE

Dieses Kapitel enthält eine alphabetisch sortierte Liste von photogrammetrie-bezogener Software mit Kosten und Beschreibung.

E.1 AGISOFT PHOTOSCAN

Professionelle Photogrammetrie-Software zur Rekonstruktion von 3D-Modellen aus Fotos, sowie zur Erstellung von Orthofotos und Oberflächenmodellen.

Lizenz: Proprietär.

Kosten: Pro Version 3'499 USD, Standard Version 179 USD. Sonderkonditionen für schulische Zwecke.

URL: `http://www.agisoft.com/`

E.2 BLENDER

Professionelle Open Source Software zur Erstellung, Modellierung, Bearbeitung und Animation von 3D-Modellen. Weitere Infos in Unterabschnitt 3.6.1.

Lizenz: GPLv2.

Kosten: Gratis.

URL: `https://www.blender.org/`

E.3 CLOUDCOMPARE

Open Source Software zum Betrachten und Verarbeiten von Meshes.

Lizenz: GPLv2.

Kosten: Gratis.

URL: `http://www.danielgm.net/cc/`

E.4 CURA

Von der Firma Ultimaker entwickelte Open Source Slicer-Software zum Aufbereiten von 3D-Modellen für den 3D-Druck.

Lizenz: AGPLv2.

Kosten: Gratis.

URL: `https://ultimaker.com/en/products/cura-software`

E.5 GPICSYNC

Open Source Software zum Geocodieren von Fotos. Weitere Infos in Abschnitt 4.2.

Lizenz: GPLv2.

Kosten: Gratis.

URL: `https://github.com/metadirective/gpicsync`

E.6 GPSBABEL

Software zum Extrahieren von GPS-Daten aus GPS-Geräten, sowie zum Konvertieren von GPS-Daten zwischen verschiedenen Formaten. Weitere Infos in Abschnitt 4.2.

Lizenz: GPLv2 or later.

Kosten: Gratis.

URL: `http://www.gpsbabel.org/`

E.7 HUGIN

Open Source Software zum Zusammenfügen (Stitching) von Panoramas sowie zum Entzerren von Fotos mittels Linsenprofilen. Weitere Infos in Abschnitt 3.2.

Lizenz: GPLv2.

Kosten: Gratis.

URL: `http://hugin.sourceforge.net/`

E.8 MENCI APS

Professionelle Photogrammetrie-Software zur Rekonstruktion von 3D-Modellen aus Fotos, sowie zur Erstellung von Orthofotos und Oberflächenmodellen.

Lizenz: Proprietär.

Kosten: Ca. 6000 USD.

URL: http://www.agisoft.com/

E.9 MESHLAB

Open Source Software zum Betrachten und Verarbeiten von Meshes. Weitere Infos in Abschnitt 3.5.

Lizenz: GPLv2.

Kosten: Gratis.

URL: http://sourceforge.net/projects/meshlab/

E.10 OPENDRONEMAP

Open Source Photogrammetrie-Pipeline zur Erstellung von Orthofotos und Oberflächenmodellen aus Luftbildern. Weitere Infos in Abschnitt 4.3.

Lizenz: Freie Software, viele unterschiedlich lizenzierte Komponenten.

Kosten: Gratis.

URL: https://github.com/OpenDroneMap/OpenDroneMap

E.11 PIX4DMAPPER PRO

Professionelle Photogrammetrie-Software zur Rekonstruktion von 3D-Modellen aus Fotos, sowie zur Erstellung von Orthofotos und Oberflächenmodellen.

Lizenz: Proprietär.

Kosten: Monatlich 320 CHF, jährlich 3'200 CHF oder einmalig 7'900 CHF. Sonderkonditionen für nichtkommerzielle Zwecke und schulische Zwecke.

URL: https://pix4d.com/

E.12 VISUALSFM

Photogrammetrie-Software zur Rekonstruktion von 3D-Modellen aus Fotos. Weitere Infos in Abschnitt 3.4 und Anhang B

Lizenz: Quelloffen aber Proprietär. Einzelne Komponenten Freie Software. Nur zu nichtkommerziellen und schulischen Zwecken einsetzbar.

Kosten: Gratis.

URL: http://ccwu.me/vsfm/

LITERATURVERZEICHNIS

[1] Blender Foundation. Sintel, 2010. URL http://archive.blender.org/blenderorg/blender-institute/durian-open-movie/index.html.

[2] Blender Foundation. Tears Of Steel, 2012. URL https://mango.blender.org/.

[3] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[4] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Towards internet-scale multi-view stereo. In *CVPR*, 2010.

[5] Michal Jancosek and Tomás Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3121–3128. IEEE, 2011.

[6] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[7] DavidG. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B%3AVISI.0000029664.99615.94.

[8] Stephen Mather. From Dobongsan to Mount Saint Helens — FOSS4Gs and OpenDroneMap. URL https://smathermather.wordpress.com/2014/09/08/from-dobongsan-to-mount-saint-helens-foss4gs-and-opendronemap/.

[9] Mathias Rothermel, Konrad Wenzel, Dieter Fritsch, and Norbert Haala. Sure: Photogrammetric surface reconstruction from imagery. In *Proceedings LC3D Workshop, Berlin*, 2012.

[10] Fabian Langguth Simon Fuhrmann and Michael Goesele. MVE - A Multi-View Reconstruction Environment. In *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage (GCH)*, 2014.

[11] Changchang Wu. VisualSFM: A Visual Structure from Motion System. URL http://ccwu.me/vsfm/.

[12] Changchang Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/~ccwu/siftgpu, 2007.

[13] Changchang Wu. VisualSFM: A Visual Structure from Motion System. 2011. URL http://www.cs.washington.edu/homes/ccwu/vsfm.

[14] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.