IPA Cambotprocessor

Dokumentation IPA

Teil 1: obligatorische Kapitel

### Vorwort

Diese Dokumentation ist im Besitz der Technische Fachhochschule Nordwestschweiz in Brugg-Windisch. Die Dokumentation beinhaltet den Ablauf der Entwicklung einer Commandline App

Die Dokumentation befasst sich zuerst mit der Aufgabenstellung und danach der Umsetzung einer Lösung zu der genannten Aufgabenstellung.

### Involvierte Personen

Kandidat: Verantwortliche Fachkraft:

Semjon Buzdin Martin Gwerder  
FHNW FHNW  
Bahnhofstrasse 6 Bahnhofstrasse 6  
+41 79 573 12 72 +41 56 202 76 81  
[semjon.buzdin@fhnw.ch](mailto:semjon.buzdin@fhnw.ch) [martin.gwerder@fhnw.ch](mailto:martin.gwerder@fhnw.ch)

Hauptexpert: Nebenexperte:

André Lichtsteiner Wing Lung Choong  
  
Hohlstrasse 355  
8004 Zürich  
+41 79 469 33 67  
andre.lichtsteiner@gmx.ch

# Inhaltsverzeichnis

# Aufgabenstellung

### Titel der Facharbeit

Der Titel der Facharbeit lautet: «cambotprocessor»

### Ausgangssituation

CamBot ist ein für diese Arbeit neu entwickelter Roboter. Er ist in der Lage eine Kamera, um einen 3D Drucker frei und repetierbar zu positionieren. Dieser Roboter soll in OctoPrint einbindbar sein und 3D-taugliche Daten sowie Hyperlapses (Zeitraffer mit Kamerabewegung) liefern, die mittels der in der Intel Realsense d435 und Meshroom (Fotogrammmetrie) gewonnen werden.

Auch wenn die Aufgabe aufbauend auf der Aufgabe von Maurice Meier ist, kann Semjon unabhängig von Maurice arbeiten. Er erhält schon vorgefertigte Bildsets auf USB-Sticks, so dass er nicht auf die Arbeit von Maurice angewiesen ist.  
  
Die IPA Startbesprechung fand am 04.03.2022 statt um 15:00 da Martin Gwerder am Montag nicht zur Verfügung stand.

### Aufbau der Dokumentation

Diese Dokumentation richtet sich nach den Vorgaben aus dem Dokument «Kriterienkatalog Standardkriterien Buzdin Semjon». Die Dokumentation ist in zwei Teile aufgeteilt. Der erste Teil «obligatorische Kapitel» beschreibt die Umgebung der Facharbeit. Der zweite Teil dokumentiert die eigentliche Arbeit. Die Gliederung dieses zweiten Teils richtet sich nach den Schritten der Projektmethode IPERKA.

### Kurzfassung

### Detaillierte Aufgabenstellung

Ziel dieser Arbeit ist es die Bilddaten von Cambot über ein Rest-Interface abzurufen, diese effizient und repetierbar auf einem SMB-Share abzulegen und mittels einer festen Meshroom Queue aufzubereiten. Dazu soll der "cambotprocessor" (ein Kommandozeilentool) geschrieben werden. Der "cambotprozessor" holt die Dateien vom "cambotmanager" via RestAPI ab und verarbeitet diese unter Inbezugnahme eines Konfigurationsfiles und den Kommandozeilen-Parameter.

Die Standard-Konfigurationsdatei, die gesucht wird, heisst "cambotprocessor.ini" und wird vom Kommandozeilenparameter -- config übersteuert. Per Default die Datei wird gesucht (1.) im aktuellen Arbeitsverzeichnis und anschliessend (wenn nicht gefunden) im Verzeichnis wo sich das Skript "cambotprocessor "befindet.

Das CLI-Programm wird wie folgt spezifiziert (wobei Kommandoozeilenparameter die Settings aus dem Konfigurationsfile ergänzen respektive übersteuern):   
usage: cambotprocessor [--config <configfile>] [--blur <schärfewert>] [--outputtype <ply|avi|stl|html>] [--meshroomqueue <spec>] [--bounding <spec>] [<Verzeichnis, Zip oder URL der Bilddateien> <Output Directory>

Die Konfigurationsdatei ist In der Konfigurationsdatei (--config; default: cambotprocessor.ini) können defaultwerte für alle Kommandozeilenparameter angegeben werden (alle in lowercase ohe die einführenden Minuszeichen). Der Konfigschlüssel für <Verzeichnis, Zip oder URL der Bilddateien> lautet "url" und der Konfigschlüssel für <Output Directory> lautet "output".

Das Zip-File ist wie folgt strukturiert: <taskname> --- snapshots --- <iso8601-timestamp>--- metadata.ini and images (png or jpg)

Die Ausgabestruktur soll folgendes Beinhalten (zusätzlich zum Inhalt des Zipfiles, welcher in das outputverzeichnis kopiert wird).   
<output-directory> --- output --- <outputtype> --- outputfiles

Das Programm macht folgendes:

* Es entpackt das ZIP-File in eine temporäre Struktur (falls notwendig).
* Bildinformationen und Bilder und in eine Meshroom-Verträgliche Struktur geschrieben (siehe oben)
* Bilder mit einem zu tiefen Blur-Wert werden aufgefiltert (umbenannt in <filename>.filtered)
* Meshroom-Queue wird gestartet - Eventuell wird ein Postprocessing gemacht (avi/stl/html) und in das <Output Directory> geschrieben.

Pflichtanforderungen

Der ganze Quellcode ist verfügbar auf GitHub unter einer Quelloffenen Lizenz, die kompatibel ist mit den verwendeten Lizenzen (Vorschlag: GPL) der eingesetzten Module.

Die Funktion ist in einem GitHub-Wiki dokumentiert welches Minimal folgende Punkte dokumentiert:

Alle Parameter des Kommandozeilen Programmes inklusive entsprechender Beispiele - Alle Konfigurationseinträge der Konfigurationsdatei (Mit Beispielfiles)

Installationsdokumentation (speziell für die Installation aller Dependencies)

Das Programm ist sauber und übersichtlich geschrieben und ausreichend dokumentiert (mindestens alle relevanten Funktionen mit pydoc).

Das Programm liefert sinnvolle Fehlermeldungen, wenn Files nicht gefunden werden, der Aufbau falsch ist oder etwas in der Queue oder im Postprocessing schiefläuft.

Auf einen CLI-Trigger (Kommandozeilenprogramm) hin holt die Software ein ZIP-Archiv von der RestAPI des CamBot-Managers ab (statische URL [latest] oder über eine ID)

Das Kommandozeilentool unterstützt Benutzername und Passwort bei der URL-Notation.

Alle Kamerabilder werden auf Unschärfe mittels eins geeigneten Programms untersucht und unscharfe Bilder verworfen. Dafür wird OpenCV verwendet. Beispielcode für die Blur-Detektion befindet sich unter https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/. - Mittels einer statischen Verarbeitungqueue wird eine PLY-Struktur generiert. - Das Tool liefert als Output mindestens obj, stl, ply und html (via ctmconv und https://github.com/omrips/viewstl o.ä.)

### Projektmethode

Ich benütze die IPERKA Methode, um dieses Projekt umzusetzen. IPERKA ist strukturiert in 6 Phasen und werden nacheinander durchgeführt. Die einzelnen Methoden können durch das Wort IPERKA abgeleitet werden: Informieren; Planen; Entscheiden; Realisieren; Kontrolle; Auswertung.

Ich habe die Methode ausgewählt da sie einen leicht zu folgenden und klaren Projektablauf darbietet und ich mich innerhalb von einzelnen Projekten innerhalb der Berufsschule mich schon mit dieser Methode bereitgemacht habe.

### Datensicherung

Die Dokumentation sowie das Programm sind auf Github gespeichert und versioniert (Link: <https://github.com/SemBuz/cambotprocessor> ).

### Technologien

Unix / Linux, Python wurden gewählt da Meshroom Unterstützung mit Python darbietet.

### Vorkenntnisse

Kleinere Freizeitarbeiten mit Python und Berufsschulische Erfahrung mit Raspbian OS.

### Vorarbeiten

Im Vorfeld dieser Arbeit wurden keine Vorarbeiten ausgeführt.

### Arbeiten in den letzten 6 Monaten

Semjon Buzdin hat in den letzten 6 Monaten an einem Flutter Projekt gearbeitet, welches eine Library für ein Framework zur Verfügung stellt. Die Aufgabe wurde zusammen mit zwei anderen Lehrlingen gemacht. Ebenfalls hat Semjon eine Probe IPA abgeschlossen

### Kommunikation zwischen Kandidaten und Experten

Während das Durchführen dieser IPA kommen viele Fragen auf in Bezug auf die IPA welche Mithilfe der Experten geklärt werden, das jeweilige Dokumentieren dieser Fragen wird hier aufgezeigt, mit den jeweiligen Antworten und während der Durchführung weiterhin up-to-date gehalten.

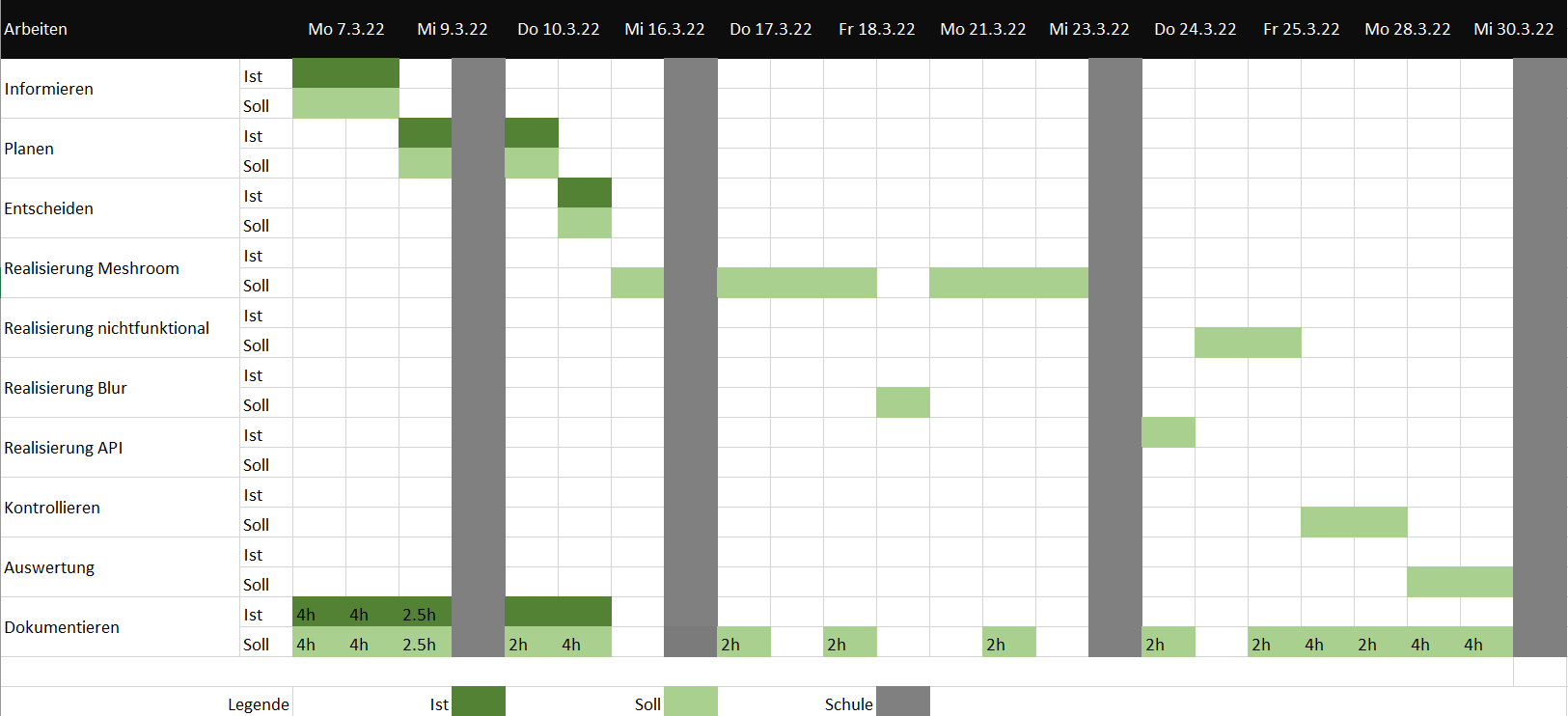
04.03.22:  
Kandidat: «Haben sie eine professionelle Benutzeranleitung?»  
Fachperson: «Ich kann eine öffentliche Studentenarbeit mit Benutzeranleitung weiterleiten.»

09.03.22

Kandidat: «Wann ist das Präsentationtermindatum?»

Hauptexperte: «19.April 2022 9:00 Uhr»

# Zeitplan



# Arbeitsjournal

Das Arbeitsjournal beinhaltet 8h Inhalte pro Tag jedoch ist meine Arbeitszeit 8.4h, die letzten 20 Minuten werden jeweils genützt, um das Arbeitsjournal des jeweiligen Tages auszufüllen, Zeitplan ausfüllen und Arbeitsjournal des nächsten Tages vorzubereiten.

### Montag, 7.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 2h | 1.75h | Dokumentation Skelett vorbereiten | Skelett der ganzen Doku aufarbeiten |  |
| 2h | 2.25h | Obligatorische Kapitel vervollständigen | Teil 1 abschliessen bis auf Abschnitt «Kurzfassung» |  |
| 4h | 3.75h | Informierens-abteil vervollständigen | Kapitel «Informieren» in Teil 2 vervollständigen bis auf weitere Kommunikation |  |
| Total | |  | | |
| 8h | 7.75h | Das Skelett aufarbeiten war kein grosser Auftrag da ich mir der Probe-IPA schon viel aufgebaut habe und dies somit nicht viel Arbeit darbietete als einfaches abschreiben und einige Fehler ausbessern.  Das Vervollständigen der Obligatorischen Kapitel war zum grösstenteils kopieren einiger Texten, jedoch gab es noch Probleme da ich die kompletten Kontaktdaten des Haupt- und Nebenexperten nicht zur Verfügung habe und diese noch benötigte.  Die grösste Arbeit der obligatorischen Kapitel ist das Erstellen des Zeitplans und abwerten wann ich wo etwa fertig sein werde, ich habe mir hier zum grösstenteils viel Zeit für das Realisieren und Planen gelassen da hier evtl. noch Probleme auftreten.  Den Nachmittag habe ich mir des Informierens gewidmet und das Kapitel im Dokument auszufüllen, hier entstand keine Hürde, wiederholt fand ich Punkte die erwähnt werden sollte und bin zufrieden mit der Anforderungsanalyse, die ich erstellte nach der Kritik, an welcher ich erhielt bei der Probe IPA. Die Letzte übrige 15 Minuten verwendete ich für Recherche wie Meshroom integriert werden sollte da dies mich besorgte | | |

Probleme: -

Fragen: -

Recherchen: <https://stackshare.io/stackups/go-vs-python-vs-rust> , <https://solaceinfotech.com/blog/comparison-python-vs-ruby-vs-golang/#:~:text=Python%20and%20Ruby%20are%20slower,contains%20a%20fully%20web%20server> , <https://www.reddit.com/r/commandline/comments/fxydic/what_is_the_ideal_language_for_clis/>

Sicherungen heute: Initiale Version der Dokumentation

### Mittwoch, 9.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 1h | 0.75h | Use-Case Diagramm erstellen | Use-Case Diagramm erstellen und dokumentieren |  |
| 1.5h | 1.5h | Klassendiagramm erstellen | Klassendiagramm erstellen und dokumentieren |  |
| 1.5h | 1.5h | 1.Expertentreffen | - |  |
| Total | |  | | |
| 4h | 3.75h | Das Erstellen des Use-Case Diagramm war relativ kurzgehalten, ich musste mich zuerst einfach in Visual Paradigm, welches ich nutze, um UMLET Diagramme zu erstellen, wieder angewöhnen.  Das Erstellen des Klassendiagramm war eine grössere Anforderung da sich die Standardqueue von Meshroom über viele Funktionen streckt und ebenfalls viele Parameter existieren, somit gab dies mehr Arbeit welches ich schon vorhersah da Meshroom so ein extensives Programm ist und nutzte die ganze 1.5 Stunden, die ich mir zur Verfügung gegeben habe  Danach hatte ich noch 15 Minuten frei vor dem Expertentreffen welche ich genützt habe, um eine mögliche Implementation von Blur Detection zu recherchieren | | |

Probleme:

Fragen: Wann ist der 3.Termin (Präsentationstermin) – 19.April

Recherchen: <https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/>

Sicherungen heute: Dokumentation welche weiterhin daran gearbeitet wurde

### Donnerstag, 10.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 3h | 3h | Erfüllung von I5 innerhalb der Dokumentation | Machbarkeit recherchieren und im Abschnitt planen Dokumentieren |  |
| 1h | 0.5h | Zeitplan erweitern | Zeitplan erweitern und granulieren nach Anforderungen von Experten |  |
| 2h | 2.5h | Plannen vervollständigen | Abschnitt Planen vervollständigen mit Erklärung von Parameter und weiterem |  |
| 2h | 1h | Entscheidungen führen und dokumentieren | Entscheidungen so weit wie möglich erschliessen und fällen und Prozess dokumentieren |  |
| Total | |  | | |
| 8h | 7h | Das Erfüllen von I5 benötigte einige Recherche, insbesondere die Quellen, die ich zuvor gefunden habe, noch genau untersuchen und ausprobieren, die brauchte einiger der Zeit auf und schlussendlich das Evaluieren wie umsetzbar die Applikation ist, doch schlussendlich wurde ich zeitlich fertig  Das Erweitern und Granulieren des Zeitplans verlief sehr schnell und benötigte viel weniger Zeit als betrachtet, die einzelnen Schritte, um den Plan zu granulieren waren ziemlich klar und somit musste ich mir nicht sehr viel überlegen  Das Vervollständigen des Planens war komplizierter da die Syntax für die Boundingbox schwer aufzufinden war, das Wiki von Meshroom ist nicht vervollständigt und somit musste ich die Logs von dem Programm untersuchen und somit konnte ich schlussendlich die Syntax finden und dokumentieren  Das Entscheiden fiel kurz da ich herausstellte das es kaum etwas zu entscheiden gab und ich somit die erklärte und etc. somit wurde ich 1h früher fertig, jedoch fühlte ich mich nicht sehr fit und beging mich Nachhause und Arbeite morgen 9h | | |

Probleme: Boundingbox Syntax finden

Fragen:

Recherchen: <https://groups.google.com/g/alicevision/c/-snieF7V4Ls> , <https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/> , <https://meshroom-manual.readthedocs.io/en/bibtex1/node-reference/nodes/SfMTransform.html>

Sicherungen heute: Fertigstellung von dem Dokument bis zur Realisierung

### Notiz 11.03.2022 und 14.03.2022

Wurde mit Covid-19 infiziert und IPA wurde für diese zwei Daten pausiert.

### Mittwoch, 16.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 4h | 4h | Realisierung Anbindung Meshroom | Meshroom via Commands ausführen |  |
| Total | |  | | |
| 4h | 4h | Ich begann mit dem Realisieren der Applikation. Ich setzte mir eine unbekannte Zeit vor da ich mir nicht ganz klar war wie viel der Workload exakt sein wird, Ich habe somit angefangen die einzelnen Schritte von Meshroom in der Struktur zu übertragen und erstmal angefangen mit der Integration der Parameter für welche ich argparse genutzt habe welche ich über diese Stackoverflow Frage gefunden habe <https://stackoverflow.com/questions/40001892/reading-named-command-arguments>. Die Integration dieser verlief fehlerfrei  Danach begann ich mit der Integration von Meshroom welche ich via den Binary exe gemacht habe welches auch die meiste Zeit heute anforderte. Die Parameter welche nötig waren, waren in der Hilfestellung der Exe Eingabe angegeben oder sonst auf dem Meshroom Wiki. Ich konnte mit den ersten 5 Schritten fertigwerden und konnte diese ausführen und die letztlichen Daten aus diesen kriegen.  Mit dem 6.Schritt habe ich angefangen aber noch nicht ausgeführt. | | |

Probleme:

Fragen:

Recherchen: <https://stackoverflow.com/questions/40001892/reading-named-command-arguments>, <https://meshroom-manual.readthedocs.io/en/latest/> , <http://filmicworlds.com/blog/command-line-photogrammetry-with-alicevision/>

Sicherungen heute: Initialer Commit von Cambotprocessor

### Donnerstag, 17.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 4h | 3.5h | Fertigstellen der Meshroom Integration | Meshroom SQueue sollte durchgeführt werden können |  |
| 2h | 1h | Config Funktionalität einfügen | Cambotprocessor Parameter sollten durch Configfile aufrüfbar sein |  |
| 2h | 2h | Realisierungsdokumentation beginnen | Struktur umschreiben und StandardQueue beschreiben |  |
| Total | |  | | |
| 8h | 6.5h | Das Fertigstellen der Meshroom Integration verlief schnell da ich mit der Hilfe eins existierenden Programms vieles kopieren konnte und anpassen musste da das existierende Programm mit Meshroom 2018.1.01 gemacht wurde. Der Prozess verlauft jetzt und wurde getestet mit Testdaten, welche zwar relativ schlecht sind, aber der Prozess nicht so lange benötigt wie bei den Meshroomtestdaten welche mehr als eine Stunde zum Durchlaufen benötigen  Das Einbauen der Config Funktionalität war sehr einfach da ich dies schon mehrmals zuvor gemacht habe und somit Code kopieren konnte.  Zur Realisierung gibt es einige Aspekte, die ich ansprechen möchte und heute nur die Standardqueue angesprochen habe und die Struktur des Projektes welche sich eigentlich aus dem Plannen sich ergeben hat.  Mit der Bonuszeit habe ich noch einige der Bewertungskriterien reviewt und angefangen über das Speichern des Outputtype zu recherchieren | | |

Probleme:

Fragen:

Recherchen: <http://filmicworlds.com/blog/command-line-photogrammetry-with-alicevision/> , <http://manpages.ubuntu.com/manpages/bionic/man1/ctmconv.1.html> , <https://pymesh.readthedocs.io/en/latest/installation.html>

Sicherungen heute: MeshroomStandardQueue fertiggestellt, Configfile Parameter integriert und Realisierungsdokumentation geupdated

### Freitag, 18.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 2h | 2.25h | Output nach outputtype anpassen | Output kann angepasst werden |  |
| 2h | 3h | Doku Realisierung schreiben | Wie Blur implementiert wird und Mögliche Erweiterungen erläutern |  |
| 4h | 3h | Blur implementieren | Blurdetection implementieren |  |
| Total | |  | | |
| 8h | 8.25h | Die Implementation des Outputtype anpassen war ziemlich kompliziert da ich ein Problem hatte eine passende Library zu finden, zuerst wollte ich anschauen, wie ich Martins vorgeschlagenes Programm nutzen kann (ctmconv) jedoch fand ich keine wirkliche Implementation. Später stiess ich auf PyMesh, welche die Methoden beinhaltet, welche ich brauche, jedoch die Implementation dieser wäre zu komplex für jeglichen Nutzer und ich selbst kriegte die Implementation nicht hin, nach einiger Recherche fand ich jedoch meshio und mit meshio war die Implementation sehr simpel.  Doku schreiben war ein wenig schwer da ich die Mathematik hinter dem Ganzen erklären wollte, jedoch diese nicht selber verstanden habe und somit diese Erklärung sehr reduzierte  Das Implementieren des Blurs verlief schnell da diese sehr simpel ist mit Opencv. | | |

Probleme:

Fragen:

Recherchen: <https://github.com/nschloe/meshio> , <https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/> , <http://manpages.ubuntu.com/manpages/bionic/man1/ctmconv.1.html>

Sicherungen heute: Outputtype Implementation, Blurdetection Branch, Doku update

### Montag, 21.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
| 2h | 1.5h | Customqueue implementieren | Customqueue Funktion einbauen |  |
| 2h | 1h | Code kommentieren | Bestehenden Code komplett kommentieren |  |
| 2h | 2h | APICommunication implementieren | APICommunication implementieren nach Anforderungen |  |
| 2h | 2h | Schnittstellen in Doku erläutern | APICommunication und Schnittstelle mit Meshroom dokumentieren |  |
| Total | |  | | |
| 8h | 6.5h | Das Implementieren der Customqueue benötigte ein bisschen Zeit da den Command selbst zu schreiben recht viel Arbeit darbietet und somit brauchte es ein bisschen bis ich diese testen konnte. Jedoch dann mit Erfolg  Da die einzelnen Meshroomschritte innerhalb der .exe ausgeführt werden gibt es hier nicht massiv viel zu kommentieren, jedoch was nötig war zum Kommentieren wurde kommentiert.  Das Implementieren der APICommunication war zuerst nur sehr kurz, doch brauchte ich einige Zeit, bis ich auch die Abstraktion implementieren konnte, da ich mir unsicher war wie dieses Aussehen sollte, jedoch nach einigem Recherchieren kam ich zu einer Lösung  Mit der Restlichen Zeit habe ich angefangen das Benutzermanual zu schreiben und Benutzermanualbeispiele zu lesen. | | |

Probleme: -

Fragen: -

Recherchen: -

Sicherungen heute: Kommentieren, APICommunication + APICommunicationTests, Doku update

### Mittwoch, 23.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
|  |  |  |  |  |
|  |  |  |  |  |
| Total | |  | | |
|  |  |  | | |

Probleme:

Fragen:

Recherchen:

Sicherungen heute:

### Donnerstag, 24.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
|  |  |  |  |  |
|  |  |  |  |  |
| Total | |  | | |
|  |  |  | | |

Probleme:

Fragen:

Recherchen:

Sicherungen heute:

### Freitag, 25.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
|  |  |  |  |  |
|  |  |  |  |  |
| Total | |  | | |
|  |  |  | | |

Probleme:

Fragen:

Recherchen:

Sicherungen heute:

### Montag, 28.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
|  |  |  |  |  |
|  |  |  |  |  |
| Total | |  | | |
|  |  |  | | |

Probleme:

Fragen:

Recherchen:

Sicherungen heute:

### Mittwoch, 30.3.2022

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Soll | Ist | Tätigkeiten | Ziele | Bemerkung |
|  |  |  |  |  |
|  |  |  |  |  |
| Total | |  | | |
|  |  |  | | |

Probleme:

Fragen:

Recherchen:

Sicherungen heute:

Teil 2: Projekt-Dokumentation

# Informieren

Dieses Abteil dokumentiert den Informieren Aspekt des Projektes und wie der Kandidat vorgegangen ist. Und Kommunikation zwischen dem Kandidaten und den einzelnen Fachpersonen

### Was ist Meshroom?

Meshroom ist eine gratis, open-source 3D Rekonstruktionssoftware, welche auf dem AliceVision Framework basiert ist.

AliceVision ist ein photogrammetrisches Framework welches 3D Rekonstruktion- und Kamera Tracking Algorithmen darbietet. Diese werden in Meshroom genützt, um aus einer Sammlung von Bildern ein 3D Model zu erstellen aus diesen genannten Bildern.

Teil des Bilderset: Ausgabe:





### Deklarierung Programmiersprache

Die Umsetzung des Programms muss eine Commandline Programm sein somit muss sollte eine Skriptsprache gewählt werden. Beliebte und gut dokumentierte Skriptsprachen, welche existieren sind:

* Python
* C
* Rust
* Ruby
* Go

Nun müssen wir einige Kriterien deklarieren, um diese Programmiersprachen abzuwägen, die Kriterien, welche ich diese Programmiersprachen abwiegen werde, sind:

* Bekanntheit mit der Sprache
* Integrationsmöglichkeit mit Meshroom
* Breite an Libraries
* Performance

Diese Kriterien wurden gewählt da:

* Die Bekanntheit mit der Skriptsprache erlaubt mir das Projekt säubere rumzusetzten und gut lesbaren Code zu erstellen
* Die Integrationsmöglichkeit mit Meshroom erlaubt mir Meshroom leichter einzubauen und ebenfalls erlaubt weitere Erweiterungsmöglichkeiten an einem späteren Zeitpunkt.
* Die Breite an Libraries ist nützlich zur Ermittlung einer Library zum erkenne von blurry Images und ebenfalls zum Datenmanagement mit der API
* Die Performance ist ein Punkt, welcher angesprochen werden soll, da das Erstellen von eines Meshroom 3d Model zum Teil mehrere Stunden anfordern kann.

Nun können wir eine Entscheidungsmatrix erstellen mit Gewichtungen, welche je nach der Wichtigkeit des Kriteriums gewählt wurden. Die Bewertung wird getroffen je nach eigener Erfahrung oder Recherche zu der relativen Programmiersprache. Bewertungen gehen von 1-3, 1 bedeutet das dieses Kriterium nicht mit vielen Mängel erfüllt wird, 3 bedeutet das dieses Kriterium komplett erfüllt wird, 2 bedeutet, dass das Kriterium erfüllt, wird mit wenigen Mängel.

Somit erfolgt die Entscheidung:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kriterien | Gewichtung | | Python | | | C | | Rust | | |
|  |  | | Bewertung | Resultat | | Bewertung | Resultat | Bewertung | | Resultat |
| Bekannth. | 2 | | 3 | 6 | | 3 | 6 | 1 | | 2 |
| Int. Mesh. | 3 | | 3 | 9 | | 1 | 3 | 2 | | 6 |
| Libraries | 2 | | 3 | 6 | | 3 | 6 | 2 | | 4 |
| Perform. | 2 | | 1 | 2 | | 2 | 4 | 3 | | 6 |
|  |  | |  | 23 | |  | 19 |  | | 18 |
| Ruby | | | | | **GO** | | | | | |
| Bewertung | | Resultat | | | Bewertung | | | | Resultat | |
| 2 | | 4 | | | 1 | | | | 2 | |
| 2 | | 6 | | | 1 | | | | 3 | |
| 2 | | 4 | | | 2 | | | | 4 | |
| 1 | | 2 | | | 3 | | | | 6 | |
|  | | 16 | | |  | | | | 15 | |

Durch dieses Abmessen der Kriterien wählte ich Python da Meshroom selbst in Python geschrieben wurde, ich schon zuvor mit Python gearbeitet habe und da Python einer der weltweit meistgenutzten Programmiersprachen ist gibt es auch viele verfügbare Libraries zur Auswahl.

### Kommunikation zwischen Fachpersonen

Während das Durchführen dieser IPA kommen viele Fragen auf welche Mithilfe der Fachpersonen geklärt werden, das jeweilige Dokumentieren dieser Fragen wird hier aufgezeigt, mit den jeweiligen Antworten und während der Durchführung weiterhin up-to-date gehalten.

09.03.2022:

Kandidat: «Ist es okay die custom Meshroomqueue als ein Stringarray von einzelnen Commandlines einzufügen?»

Fachperson: «Ja es ist okay dies so einzubauen»

Kandidat: «Ist es so richtig, dass die Commandline parameter optional sind und somit eine Eingabe wie cambotprocessor –config [config directory] funktionieren sollte?»

Fachperson: «Ja dies ist so richtig»

17.03.2022:

Kandidat: «Für die Konvertierung der Bilder was sollte ich nutzen?»

Fachperson: «Ein CLS Tool wie ctmconv»

21.03.2022:

Kandidat: «Was ist mit Workflow unter I1 gemeint?»

Fachperson: «Der ganze Prozess welches anfängt bei der API und endet bei der Ausgabe des 3d Model.»

Kandidat: «Was ist mit «Stimmige Schichtentrennungen» und «Sprechende Namensgebung» in I6 gemeint?»

Fachperson: «Schichtentrennung ist sinnvoll aufgebaut und vorgezeigt und Namensgebung ergibt Sinn innerhalb des Projekts»

### Anforderungsanalyse

Nach der detaillierten Aufgabenstellung sind einige Anforderungen ersichtbar welche hier aufgelistet sind. Diese Anforderung sind jeweils unterteilt in 3 Kategorien von funktionalen und nicht funktionalen Anforderungen. Den einzelnen Kategorien wurden auch IDs zugewiesen, Kategorien und ID sind wie folgt:

* «ANF-M-#» Anforderungen welche der Nutzer Meshroom nützt
* «ANF-A-#» Anforderungen welche der Nutzer mit der API kommuniziert
* «NANF-G-#» Nicht Funktionale Anforderungen über Dokumente in GitHub und Code

|  |  |
| --- | --- |
| Anforderung | Beschreibung |
| ANF-M-1 | Programm erstellt mit der Standard Meshroom Queue ein 3d Model aus Bildern indem alle nötigen Parameter angegeben werden und das Programm ausgeführt wird |
| ANF-M-2 | Programm filtert blurry Bilder raus und nutzt diese nicht für Meshroom mittels interner Methode. (Wird automatisch ausgeführt) |
| ANF-M-3 | Programm liefert sinnvolle Fehlermeldungen welches Fehleingaben leicht erkennbar macht |
| ANF-M-4 | Config-File wird eingelesen und Parameter beachtet |
| ANF-M-5 | Custom Queue kann angefordert werden indem parameter --meshroomqueue eine Queue eingegeben wird |
| ANF-M-6 | Bounding-Box kann angegeben werden und begrenzt das 3d-Model und entfernt Abschnitte ausserhalb dieser Box indem mit --bounding eine Boundingbox angegeben wird |
| ANF-M-7 | Ausgabe kann in verschiedenen Datentypen gespeichert werden via –outputtype |

|  |  |
| --- | --- |
| Anforderung | Beschreibung |
| ANF-A-1 | Nutzer kann Bilddaten von der API herunterladen, indem ein Command zum Downloaden des Directorys angegeben wird |
| ANF-A-2 | Bilddaten sind korrekt und können für Meshroom genützt werden |

|  |  |
| --- | --- |
| Anforderung | Beschreibung |
| NANF-G-1 | Code ist leserlich und dokumentiert |
| NANF-G-2 | GitHub Wiki erklärt den Aufbau des Programmes verständlich |
| NANF-G-3 | Benutzermanual ist verständlich und führt Nutzer problemlos zu dem gewollten Ergebnis |
| NANF-G-4 | Das GitHub Repo wurde sauber geführt |

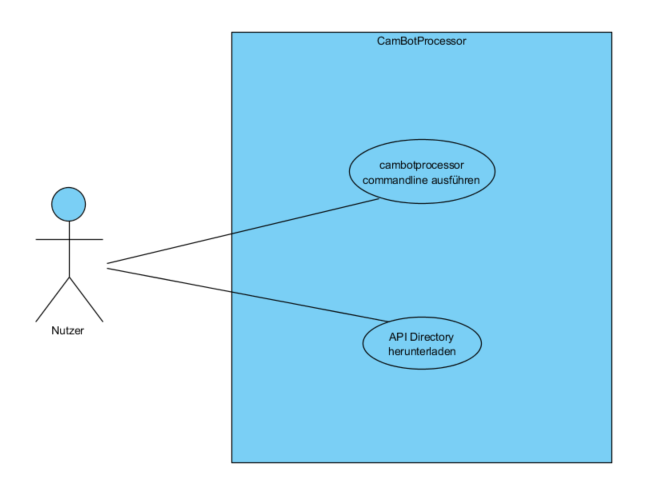
# Planen

### Use-Case Diagramm

Der Nutzer hat jeweils nicht sehr viele einzelne Eingaben mit dem Tool selbst da dies nur ein CLS Tool ist, somit ist das Use-Case Diagramm ziemlich limitiert.

Die Funktionen, welche der Nutzer primär ansteuern kann, ist cambotprocessor welche viel Anpassungsfähigkeit darbietet aber diese nicht innerhalb des Programms definiert wird, sondern ausserhalb mit Parameter und config-datei. Ebenfalls kann der Nutzer jeweilige Bildsets über die CamBotManagerAPI herunterladen.

Somit befindet sich viele Aspekte der Nutzerinteraktion ausserhalb des definierten Project-Scope.



### Klassendiagramm

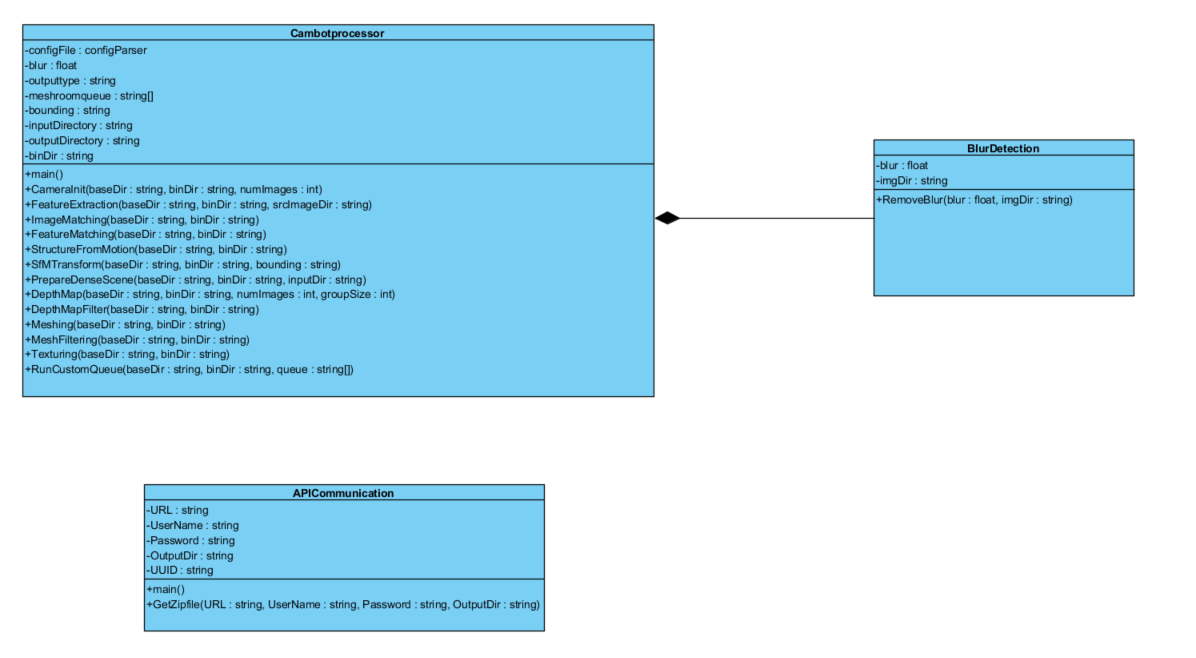
Um das Klassendiagramm zu erstellen, müssen wir zuerst die Applikation in einzelne Klassen einteilen. Die Funktionen des Projektes lassen sich leicht in 3 Teile einordnen:

* Meshroom ausführen
* Bilder herausfiltern
* API-Interaktion

Aus dieser Analyse entstehen schlussendlich 3 Klassen.

* Cambotprocessor: Hauptklasse welche auch aufgerufen wird beim Ausführen der CLS. Beinhaltet alle möglichen Schritte der Standard Meshroom Queue und das Ausführen der Custom Queue.
* BlurDetection: Beinhaltet Methoden zum Herausfiltern von blurry Images.
* APICommunication: Beinhaltet Methoden, welche zur Kommunikation mit der API dienen

Die Klasse welche zuständig ist zur Kommunikation mit der API ist nicht verbunden mit der Cambotprocessor Hauptklasse da die APICommunication separat mit einem anderem Command ausgeführt wird.



### Integration von Meshroom

Es war schnell eine Frage wie Meshroom in ein Programm eingebaut werden soll. Die Idee war es zuerst eine Library zu integrieren, jedoch existiert so keine die ich auffinden konnte innerhalb von Python. Nach einiger Weile Recherche konnte ich einige Ansätze für Integrationen von Meshroom in Python finden.

Diese Ansätze nützen jedoch nicht eine Meshroom Library, sondern führen Meshroom über Commandlines aus, was für diese Applikation gut funktionieren sollte. Somit werde ich den Ansatz, den ich online gefunden habe, anwenden und für die aktuelle funktionierende Version von Meshroom (Meshroom-2021.1.0).

Mit diesem Ansatz führt das Programm eine .exe aus mit dem angemessenen Parameter für den einzelnen Schritt. Als Beispiel:

/aliceVision\_cameraInit.exe –defaultview 45.0 –verboselevel info –sensorDatabase –allowSingleView 1 –imageFolder ./[jeweilige directory] –output ./cameraInit.sfm

Quellen welche ich genützt habe für diesen Erschloss:

<https://www.reddit.com/r/photogrammetry/comments/mpme1v/using_meshroom_as_a_lib_in_python/>

<https://github.com/alicevision/meshroom/blob/develop/bin/meshroom_batch>

<https://meshroom-manual.readthedocs.io/en/latest/feature-documentation/cmd/photogrammetry.html>

### Machbarkeit des Auftrags

Die Applikation bietet einige Schnittstellen an welche verschiedenen Integrationen genutzt werden sollten. Dies könnte zu einigen Problemen führen, somit wurde zuerst untersucht, wie umsetzbar all die Aspekte des Programms sind und wie diese umgesetzt werden sollten.

Einzelne Schnittstellen:

* Cambotprocessor-Meshroom (etabliert in Abschnitt Integration von Meshroom)
* Cambotprocessor-CambotManager (API)
* Cambotprocessor-BlurDetection

Die Schwierigkeit der Verbindung mit der API ist nahezu trivial da Python schon eine integrierte Library mit der Library «requests» anbietet welche leicht zu Nutzen ist und ich schon in einem vorigen Projekt genützt habe, die Integration mit der API sollte also nicht mehr als 2h benötigen.

Seitens Blurdetection gibt es mit Opencv schon Methoden eine Blurdetection einzubauen welche ziemlich simpel ist und nicht mehr als 30 Zeilen Code beinhaltet. Somit sollte dies auch nicht mehr als einen Halbtag Arbeit anfordern.

URL Blurdetection Integration: <https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/>

Die Schnittstelle von Meshroom zu dem Cambotprocessor wurde schon etabliert in Integration von Meshroom und ist mit der Standardqueue sicher kein grösseres Problem dies einzubauen jedoch wird dies einige Zeit benötigen (Schätzungsweise 2 Tage). Die schwierigsten Aspekte der Applikation werden jedoch Integration von einer custom Meshroomqueue und Boundingbox sein.

Im Fall der custom Meshroomqueue habe ich schon Herrn Gwerder angefragt, wie eine mögliche Integration dieser aussehen könnte. Diese würde so funktionieren das der Nutzer ein Stringarray aus einzelnen vollständigen Commands eingeben würde und diese einfach ausgeführt werden würde. Diese Weise die custom Meshroomqueue einzubauen ist für mich gesehen ziemlich einfach, die eingabe einer custom Meshroomqueue wird einfach mühsam für den Nutzer sein, jedoch jeden einzelnen möglichen Meshroom Schritt einzubauen würde das Workload des Projektes sprengen da es insgesamt in Meshroom-2021.1.0 über 60 Schritte gibt.

Im Falle der Boundingbox gibt es hier einen designierten Meshroom Schritt, der Schritt «SfMTransform» welche die Boundingbox implementiert, jedoch ist die Eingabe für diese ziemlich spezifisch und muss sicher mit experimentiert werden damit ich dem Nutzer eine vorgeschlagene Eingabe darbieten kann. Jedoch sollte die Integration und testen dieser innerhalb eines halben Tages bis einen ganzen Tag gut möglich sein.

Diesen Abschnitt Machbarkeit des Auftrags ist für die Bewertung für Bewertungskriterium I5. Commit: 87676565d2c9042508c4e5eb2e6eb9aab463f0b4

### Nutzen Parameter

In dem CLS-Trigger cambotprocessor gibt es einige Parameter, welche beachtet werden müssen zur Ausführung des Programms, diese sind:

* --config
* --blur
* --outputtype
* --meshroomqueue
* --bounding
* --input
* --output
* --binary

Diese einzelnen Parameter haben verschiedene InputsSyntax und jeweiligen Nutzer, welcher hier aufgeführt ist:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | InputSyntax | Beispiel | Beschreibung |
| --config | string (Filedirectory) | --config ./cambotprocessor.ini | Beinhaltet Eingaben für alle anderen Parameter damit diese gespeichert werden können (optional) |
| --blur | float | --blur 100 | Bestimmt Grenze, ab welcher ein Bild aus der Bilddirectory entfernt werden sollte, da ein Bild zu blurry ist und zu unklar, um zu erkennen. |
| --outputtype | string (fileending) | --outputtype stl | Bestimmt Fileending in welche das 3d Model gespeichert werden sollte (standard als Obj). Ohne angegebenen outputtype wird ein texturiertes Objekt ausgegeben. (optional) |
| --meshroomqueue | string[] | --meshroomqueue /aliceVision\_cameraInit.exe –defaultFieldOfView 45 –verboseLevel info –allowSingleView 1 –imageFolder ./Bilder –output cameraInit.sfm , /aliceVision\_featureExtraction.exe –describerTypes sift –forceCpuExtraction True –verboseLevel info –rangeStart 0 –rangeSize 100 –input camerainit.sfm –output ./ | Beinhaltet gesamte Commandlinequery zum Ausführen einer komplett modifizierbaren Meshroom-queue  (optional) |
| --bounding | string (X,Y,Z,Euler,Scale) | --bounding 10,10,10,0,0,0,1,1,1 | Erstellt eine Boundingbox welche Abschnitte ausserhalb der definierten Box entfernt mit jeweils 5 Parameter. X: Länge der Box auf der X-Achse, Y: Länge der Box auf der Y-Achse, Z: Länge der Box auf der Z-Achse, Euler: Rotation der Box in Euler, S: Skalierung der Box in XYZ (optional) |
| --input | string (Filedirectory) | --input ./bilder | Directory von Bildern welche für Meshroomqueue genutzt werden |
| --output | string (Filedirectory) | --output ./result | Directory in welchen das Endprodukt gespeichert wird (3d Model + Texturing Datei) |
| --binary | string (Filedirectory) | --binary ./aliceVision/bin | Directory der Binary von Meshroom zum Ausführen von Meshroomfunktionen |

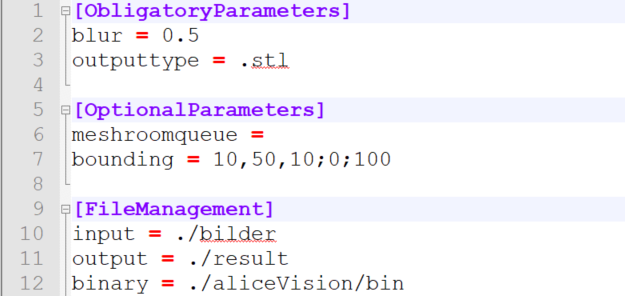
--binary wurde noch im Nachhinein hinzugefügt da diese auch nötig ist zum Ausführen des Meshroom Prozesses.

### Aufbau Config-Datei

Die Configdatei dient dazu da, dass die Parameter abgespeichert bleiben. Die Config-Datei muss auch strukturiert richtig strukturiert sein, die Configdatei ist in 3 Abteile geordnet:

* ObligatoryParameters: beinhaltet Parameter welche nötig für das Ausführen des Programmes ist
* OptionalParameters: beinhaltet optionale Parameter
* FileManagement: beinhaltet alle Parameter, welche ein Directory sind

Somit sieht eine vollständige Configdatei so aus:



# Entscheiden

Da die Detaillierte Aufgabenstellung und Ausgangslage nahezu alle Aspekte des Programms beinhalten und klar entscheiden gibt es wenig zu besprechen. Es gibt einige Aspekte die verschieden angegangen werden könnten, diese jedoch wie schon gesagt definiert sind.

Das Projekt könnte in verschiedenen Umsetzungsmethoden erfüllt werden. Wie mit einem UI oder in einer Konsolenapplikation anstatt einer Commandline Applikation.

Diese einzelnen Umsetzungsmethoden würden verschiedene Vorteile aber auch Nachteile bringen:

Pros:

* Commandline Applikation:
  + Performanceweise am schnellsten
  + Ermöglicht leichte Eingabe von Parameter
* Konsolen Applikation:
  + Mehr Übersicht über Parametereingabe
* UI Applikation:
  + Mehr Übersicht über Parametereingabe
  + Klares Feedback für den Nutzer

Cons:

* Commandline Applikation:
  + Parametereingabe ohne klare Fehlerüberprüfung
* Konsolen Applikation:
  + Parametereingabe ist langsam mit Fehlerüberprüfung
* UI Applikation:
  + Performanceweise am langsamsten und Prozess braucht lange
  + Würde Umfang des Projekts sprengen

Nun würde es Sinn machen eine Entscheidungsmatrix zu führen sowie zuvor im Abschnitt Deklarierung Programmiersprache, um diese 3 Varianten auszuwerten, jedoch wurde schon klar deklariert das die Applikation im Rahmen einer Commandline Applikation realisiert werden sollte. Somit ist dieser Abschnitt von IPERKA sehr kurz gehalten da es hier nicht viel zum Entscheiden gibt.

# Realisierung

### Struktur

Das ganze Programm ist klein darum beinhaltet das ganze Projekt nur 4 Dateien.

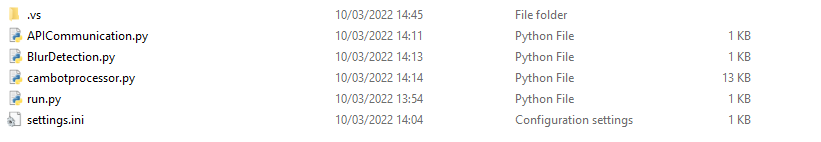
* Cambotprocessor.py
* APICommunication.py
* BlurDetection.py
* Settings.ini

Um das Ganze zu starten, nutzt man python Cambotprocessor.py [parameter], Cambotprocessor ist auch die Datei, welche grösstenteils der Logik beinhaltet. Cambotprocessor läuft alles was mit Meshroom zu tun hat durch, die StandardQueue und Customqueue.

BlurDetection beinhaltet den ganzen Code, welcher die Inputdirectory nach blurry images durchsucht und diese löscht, wie diese Funktioniert wird erläutert in «Wie funktioniert Blurdetection?».

APICommunication.py beinhaltet die Interaktionen mit der API, jedoch wird diese extern ausgeführt mit «python APICommunication –output [outputdirectory] –id [id]» Die Kommunikation mit der API ist nicht sehr extensiv somit ist diese Datei ziemlich klein gehalten.

(run.py existiert nur um das Ausführen des cambotprocessors schneller zu machen welches während des Realisierens testen schneller ermöglichte)



### StandardQueue Prozess

Der Standardqueue verfügt über 11 Schritte welche jeweiligen Inputs benötigen und verschiedene Outputs besitzen, diese Schritte können sich auch über verschiedene Versionen von Meshroom ändern, für cambotprocessor wurde jedoch Meshroom 2021.1.0 genutzt.

Die einzelnen Schritte sind:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Input | Output | Beschreibung |
| Camerainit | Bilderdirectory | sfm Datei der Bilddaten | Initialisiert Bilder-Metadata und Sensoren Information |
| Featureextraction | sfm Datei der Bilddaten | Directory mit feature Daten wie Kameraposition | Erkennt Merkmale in Bildern und extrahiert diese |
| Imagematching | sfm Datei der Bilddaten und Directory mit feature Daten wie Kameraposition | Txt mit Übereinstimmungen von Bildern | Erkennt Ähnlichkeiten in Bildern und stimmt diese überein |
| Featurematching | Txt mit Übereinstimmungen von Bildern, sfm Datei der Bilddaten und Directory mit feature Daten wie Kameraposition | Txt mit Übereinstimmungen von Bildernpositionen | Nimmt Merkmale in Bildern und stimmt diese mit Merkmalen anderen Bildern überein |
| Structure\_  from\_motion | Txt mit Übereinstimmungen von Bildern, Txt mit Übereinstimmungen von Bildernpositionen, sfm Datei der Bilddaten und Directory mit feature Daten wie Kameraposition | Sfm Datei mit einer Zusammenfassung der jeweiligen Daten und verschiedene Diagramme | Kreiert eine 3D Struktur aus Inputdaten |
| Prepare\_dencescene | Sfm Datei mit einer Zusammenfassung der jeweiligen Daten | Aufarbeitung der Bilderdaten | Entfernt Bilderverzerrungen und erzeugt EXR Bilder |
| Depthmap | Sfm Datei mit einer Zusammenfassung der jeweiligen Daten und Aufarbeitung der Bilddaten | Bilderdaten aufarbeitet mit depthdata | Erzeugt Tiefendaten für jeden Pixel der Bilder |
| Depthmapfilter | Sfm Datei mit einer Zusammenfassung der jeweiligen Daten und Bilderdaten aufarbeitet mit depthdata | Bilderdaten aufarbeitet mit depthdata und grayscale depthmap | Die ursprünglichen Tiefendateb sind nicht konsistent. Bestimmte Tiefendaten behaupten, dass sie Bereiche sehen, die von anderen Tiefenkarten verdeckt werden. Der DepthMapFilter-Schritt isoliert diese Bereiche und erzwingt Tiefenkonsistenz. |
| Meshing | Sfm Datei mit einer Zusammenfassung der jeweiligen Daten und Bilderdaten aufarbeitet mit depthdata und grayscale depthmap | Mesh der Daten | Generiert Mesh aus Tiefendaten und 3D Struktur erstellt in «Structure\_from\_motion» |
| Meshfiltering | Mesh der Daten | Korrektur des Mesh | Filtert Aspekte des Mesh, harte Kanten und etc. |
| texturing | Korrektur des Mesh | Textur und fertiggestelltes Mesh | Erstellt UV für den Mesh |

Diese sind die Schritte für die Standardqueue, die ganze Meshroom Queue und alle zusätzlich verfügbaren Schritte sind verfügbar in der Meshroom Wiki (<https://meshroom-manual.readthedocs.io/en/v2019.2/index.html>).

### Datenkonversion

Die Datenkonversion von der Standardausgabe von Meshroom (.obj) zu anderen Datenformaten wird durch die Meshio Library durchgeführt (<https://github.com/nschloe/meshio>). Da nicht alle Formate Texturing unterstützen ist Texturing nur für das Standardformat (.obj) verfügbar. Alle verfügbaren Formate sind:

* Abaqus (.inp)
* ANSYS msh (.msh)
* AVS-UCD (.avs)
* CGNS (.cgns)
* DOLFIN XML (.xml)
* Exodus (.e)
* FLAC3D (.f3grid)
* H5M (.h5m)
* Kratos (.mdpa)
* Medit (.mesh)
* MED/Salome (.med)
* Nastran (.bdf, .fem, .nas)
* Netgen (.vol)
* Neuroglancer precomputed format (.msh)
* OBJ (.obj)
* OFF (.off)
* PERMAS (.post)
* PLY (.ply)
* STL (.stl)
* SVG (.svg)
* SU2 (.su2)
* UGRID (.ugrid)
* VTK (.vtk)
* VTU (.vtu)
* WKT (.wkt)
* XDMF (.xdmf, .xmf)

### Wie funktioniert Blurdetection?

Durch Opencv existiert einen Algorithmus der Blur erkennt innerhalb von einem Bild. Dieser funktioniert mit einem [Laplace-Operator](https://de.wikipedia.org/wiki/Laplace-Operator), um Intensitätveränderungen in einem Bild zu erkennen welche viel Vorkommen in einem blurry Bild da das Bild so verschwommen ist. Ebenfalls wird der Algorithmus genutzt, um Ecken in einem Bild zu erkennen, ein nicht geblurrtes Bild enthält viele Ecken, während ein blurry Bild wenige enthält. Das Bild wird zuerst in eine grayscale Version des Bildes versetzt damit der Algorithmus seine Wertung exakter ausführen kann.

Durch diese Erkennungen wird jedem Bild eine Wertung gegeben je nach der Anzahl Erkennung der Algorithmus macht innerhalb einer Bild. Somit heisst hohe Wertung = sehr blurry. Der Nutzer kann somit eine Wertung für den Blur geben welcher desto höher desto toleranter zu blurry Images ist. Achtung: Jedes Bild hat einen Blur-Wert, somit sollte das Minimum nicht nahe bei 0 sein bei der Eingabe.

Normales Bild: Blurry Bild:

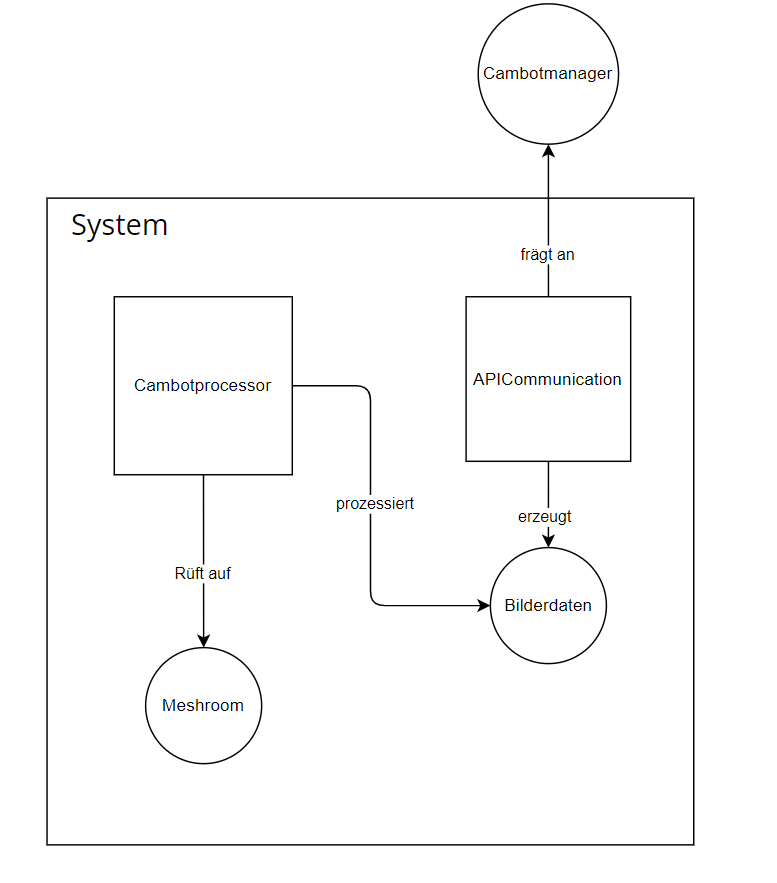


### Schnittstellen

Die Applikation beinhaltet 2 Schnittstellen. Eine Schnittstelle ist zwischen dem Cambotprocessor und dem Cambotmanager welche über eine API überbrückt wird. Die andere Schnittstelle ist zwischen dem Cambotprocessor und Meshroom.

Die API-Brücke beinhaltet einen Call, der Call wird durchgeführt durch eine andere Commandline innerhalb des cambotprocessors. Durch “python APICommunication.py –URL [URL der API] –id [custom ID] –outputdirectory [Directory mit URL output]” kann der Cambotmanager aufgerufen werden und die einzelnen Bilderdaten können werden. Die einzelnen Parameter sind:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | InputSyntax | Beispiel | Beschreibung |
| --URL | string (URL) | --URL <http://www.cambotmanager.ch/api> | URL der API des cambotmanagers |
| --id | string | --id M1l41K5234! | UUID eines Inventoryitems, das Inventoryitem unter der genannten Id wird gepullt |
| --outputdirectory | string (Filedirectory) | --outputdirectory ./inventory/ | Directory wo Bilderdaten gespeichert werden, später als –input bei dem Cambotprocessor Command nutzen |



Die Schnittstelle zwischen dem Cambotprocessor und Meshroom wird überbrückt via Commandlines. Meshroom beinhaltet einzelne .exe files welche die einzelnen steps beinhaltet mit ihrem jeweiligen input und output. Somit können wir per Python die exes ausführen und den Standardqueue durchführen. Hier ist noch eine Grafik zum Erläutern der Schnittstellen.

### Mögliche Erweiterungen

Die Applikation ist sehr extensiv jedoch beinhaltet die Applikation immer noch Erweiterungsmöglichkeiten. Hier ist eine Auflistung von verschiedenen Erweiterungsmöglichkeiten:

* Nutzer kann via APICommunication InventoryItems anfordern
* Eingabe der Custom Meshroomqueue überarbeiten, evtl. via nur Eingabe der einzelnen Schritte
* Weitere Fotogrammmetrie Services anhängen, welche die Bilderstruktur unterstützen

Diese Erweiterungen wurden aus verschiedenen Gründen nicht eingebaut, sei es das die Erweiterung den Projektumfang sprengt wie bei der verbesserten Custom Meshroomqueue da Meshroom 60+ mögliche Schritte beinhaltet. Oder einfach nicht sehr viel Sinn ergeben wie die weiteren Services anhängen da diese oft andere Eingaben benötigen und auch schon ein existierendes UI besitzen welches ein Nutzer nützen könnte.

# Kontrollieren

# Auswertung

# Quellenangaben

### Internetquellen

### Abbildungsverzeichnis

### Tabellenverzeichnis

# Glossar