

# Dokumentation für pFlow-EdgeDetector

## 1. Zielgruppe

Die Zielgruppe für diese Dokumentation sind Entwickler\*innen und Interessent\*innen.

## 2. Ordnerstruktur

In diesem Abschnitt dient dazu, um sich einen Überblick über die Ordnerstruktur zu verschaffen.

### **pFlow-EdgeDetector**

+---doc	Dokumentationen
+---experiments	durchgeführte Experimente
+---pFlowGRID	Bachelorarbeit von Nik Steinbrügge
+---training_images	Test-Bilder
+---imageCleaner.py	Methoden für Bildverarbeitung
+---main.py	Ergebnis dieser Bachelorarbeit
\---polygon.py	Klasse für Polygone

### 2. 1 Ordner: doc

In diesem Ordner befinden sich die Dokumentationen (Bachelorarbeit ([BA.pdf](#)), Dokumentation für Entwickler\*innen, Installations- sowie Benutzeranleitung) für pFlow-EdgeDetector.

### 2. 2 Ordner: experiments

Im Ordner `experiments` finden sich alle “Experimente”, die im Laufe dieser Bachelorarbeit gemacht wurden wieder.

Als Namenskonvention: `e01_beautiful_experiment.py`

<code>e</code>	steht für Experiment
<code>01</code>	ist die fortlaufende Nummer
<code>beautiful_experiment</code>	ist ein beschreibender Name, der das Experiment gut wiedergibt
<code>.py</code>	Dateiendung für Python

Dateinamen müssen mit einem Buchstaben (z. B. “e”) anfangen, wenn deren main-Methode in einer anderen Datei aufgerufen werden soll.

Die Methoden der `imageAnalyzer.py` wird in den Experimenten benutzt, um dessen (Zwischen-) Ergebnisse darzustellen.

## 2. 3 Ordner: pFlowGRID

pFlowGRID ist das Projekt, welches Nik Steinbrügge im Laufe seiner Bachelorarbeit erstellt hat. Im Ordner `doc` befindet sich `BA.pdf`, welche die Bachelorarbeit von Nik Steinbrügge ist.

```
+---pFlowGRID
|   +---doc                Bachelorarbeit von Nik Steinbrügge
|   +---floor_plans        Test-Bilder für Grundrisse
|   +---performance_tests
|   \---tmp
\ main.py
```

## 2. 4 Ordner: training\_images

```
+---training_images          Test-Bilder
|   +---floor_plan
|   |   +---htwg_building_A
|   |   +---htwg_building_O
|   |   \---Zeitschrift_Zu_meinen_Hausbau-Entwuerfen-Christian_Musel
|   +---photos
|   \---simplified_floor_plan
```

Im Unterordner `floor_plan` befinden sich unter anderem die Grundrisspläne der HTWG vom A- sowie O-Gebäude und Grundrisspläne von Christian Musel. Diese wurden aus den PDF-Dateien ausgeschnitten und als .png abgespeichert.

Im Unterordner `photos` sind Fotos enthalten, die für die Experimente `e02_transform_images.py` bis `e05_pointoperations.py` verwendet wurden.

Im Unterordner `simplified_floor_plan` wurde Bilder mit verschiedenen Komplexitätsgrade des Grundrisses vom Erdgeschoss des O-Gebäudes (`O_0.PNG`) abgelegt.

Quelle für `htwg_building_A`:

<https://moodle.htwg-konstanz.de/moodle/mod/resource/view.php?id=148848>  
[https://moodle.htwg-konstanz.de/moodle/pluginfile.php/237522/mod\\_resource/content/4/HK-AS-FL-GR-GA-GS-001.pdf](https://moodle.htwg-konstanz.de/moodle/pluginfile.php/237522/mod_resource/content/4/HK-AS-FL-GR-GA-GS-001.pdf)

Quelle für `htwg_building_O`:

<https://moodle.htwg-konstanz.de/moodle/mod/resource/view.php?id=148857>  
[https://moodle.htwg-konstanz.de/moodle/pluginfile.php/237531/mod\\_resource/content/2/HTWG-AS-FL-GR-GO-GS-001.pdf](https://moodle.htwg-konstanz.de/moodle/pluginfile.php/237531/mod_resource/content/2/HTWG-AS-FL-GR-GO-GS-001.pdf)

Quelle für `Zu_meinen_Hausbau-Entwuerfen-Christian_Musel`:

Suchbegriff: Hausbau; Suchmaschine: HTWG Bibliothek

<http://www.digizeitschriften.de/dms/img/?PID=urn:nbn:de:bsz:16-diglit-85387|log00010&physid=phys00077#navi>

Hinweis: Sollte die Software kommerziell vertrieben werden, muss davor der Ordner entfernt werden, da dies sonst gegen die Nutzungsbedingung verstößt:

<http://www.digizeitschriften.de/download/urn:nbn:de:bsz:16-diglit-85387/urn:nbn:de:bsz:16-diglit-85387-log00010.pdf>

## 3. Entscheidungen

In diesem Kapitel werden die getroffenen Entscheidungen dokumentiert.

### 3. 1 Auswahl Programmiersprache

Als Programmiersprache wurde Python gewählt, da dieser Bachelorarbeit auf der Bachelorarbeit von Nik Steinbrügge (siehe Ordnerpfad: `./pFlow-EdgeDetector/pFlowGRID/doc/BA.pdf`) aufbaut und schon dort Python verwendet wurde.

Es ist zudem eine einfache und intuitive Programmiersprache. Außerdem gibt es eine Vielzahl von Bibliotheken für Python, welche frei zugänglich sind.

#### **Weiterführende Literatur:**

Magnus Lie Hetland - *Beginning Python* - 2017 <sup>1</sup>

J. Burton Browning und Marty Alchin - *ProPython* - 2014 <sup>1</sup>

Sunil Kapil - *Clean Python* - 2019 <sup>1</sup>

Johannes Hubertz - *Softwaretests mit Python* - 2016 <sup>1</sup>

PEP 20 - *The Zen of Python* - <https://www.python.org/dev/peps/pep-0020/>

### 3. 2 Auswahl der geeigneten Bibliothek

Die Bibliothek matplotlib wurde für das Analysieren von Bildern ausgewählt. Für die Bildverarbeitung kamen PIL und OpenCV infrage. Allerdings wurde aufgrund der folgenden Gründe sich dazu entschlossen, OpenCV zu verwenden:

- OpenCV ist schneller als PIL
- Bibliothek sollte im besten Fall, verschiedene Filter zur Verfügung stellen, da der Schwerpunkt dieser Bachelorarbeit bei Filter liegt.
- OpenCV ist zwar eine Computer Vision-Bibliothek. Langfristig kann diese Vorteile mit sich bringen, weil sie in der weiteren Entwicklung eingesetzt werden kann. (z. B. 3D-Entwicklung: Wenn mittels pFlow eine Personensimulation über ein ganzes Gebäude mit mehreren Stockwerken durchgeführt werden soll.)

#### **Weiterführende Literatur:**

Literatur, die bei der Entscheidungsfindung unterstützte:

<https://www.ubuntupit.com/best-python-libraries-and-packages-for-beginners/>

[https://www.reddit.com/r/Python/comments/4u7glu/pillow\\_vs\\_opencv/](https://www.reddit.com/r/Python/comments/4u7glu/pillow_vs_opencv/)

<https://towardsdatascience.com/image-processing-opencv-vs-pil-a26e9923cdf3>

Literatur für PIL:

<https://pillow.readthedocs.io/en/latest/handbook/overview.html#image-processing>

Literatur für OpenCV:

[https://docs.opencv.org/3.4/d7/da8/tutorial\\_table\\_of\\_content\\_imgproc.html](https://docs.opencv.org/3.4/d7/da8/tutorial_table_of_content_imgproc.html)

<https://pypi.org/project/opencv-python/>

<https://www.youtube.com/watch?v=oXlwWbU8l2o>

---

<sup>1</sup> Als E-Book in der HTWG-Bibliothek verfügbar.

### 3. 3 Auswahlkriterien für Eingabe-Bild

Im Ordner "training\_images" sind alle verwendeten Bilder für dieses Projekt zu finden. Das Eingabe-Bild soll ein vereinfachtes Grundrissbild sein, damit `main.py` von `pFlow_EdgeDetector` funktioniert. Damit sind solche klaren und simplen Grundbilder gemeint, die die folgenden Bedingungen erfüllen:

- Es darf sich auf dem Bild nur ein Grundriss von einem Gebäude befinden, da ansonsten die Erstellung der Polygone in `pFlow` nicht korrekt wiedergegeben wird.
- Es dürfen keine Türen, Treppen und Hindernisse (= Wohnelemente wie z. B. Sofa, Regale, Kühlschrank etc.) im Bild enthalten sein.
- Es dürfen keine Beschriftungen (also: kein Text und Sprechblasen, keine Schilder mit Notausgang etc.) eingezeichnet sein.
- Die Ecken müssen mit jeweils zwei Linien, die im besten Fall in einem 90 Grad Winkel dargestellt werden, um die Kanten sowie Ecken entsprechend detektieren zu können. Dies ist z. B. nicht der Fall, wenn das Bild schief eingescannt wurde. Dann müsste die Schiefe des Bild vorher korrigiert werden, sodass die eben genannte Bedingung gegeben ist.

## 4. Wichtigste Python-Scripte

In diesem Kapitel werden die wichtigsten Python-Scripte von `pFlow-EdgeDetector` vorgestellt.

### 4. 1 `polygon.py`

Die Klasse `Polygon` ist die Schnittstelle zwischen `pFlowGRID` und `pFlow-EdgeDetector`.

Die Methoden, die von Nik Steinbrügge übernommen und abgeändert, wurden in der Dokumentation mit "Origin Method" sowie den Verweis auf den original Methodennamen von `pFlowGRID` gekennzeichnet.

Die Polygonzüge eines Bildes werden in einer ".nsv"-Datei gespeichert. Dabei steht ".nsv" für "nik save" und wird von `pFlowGRID` vorgegeben. Diese Datei kann mittels Texteditor geöffnet werden. Welche Daten diese Datei enthält, wird anhand vom Ergebnis des Experiments `e01_create_and_save_polygons_for_pFlow.py` erklärt:

1	2
2	-
3	4
4	100 100
5	100 500
6	800 500
7	800 100
8	3
9	120 150
10	130 110
11	125 170
12	4
13	600 270
14	600 350
15	650 350
16	650 270
17	

Zeile 1: Anzahl innere Polygone

Zeile 2: -

Zeile 3: Anzahl Knoten des äußeren Polygons

Zeile 4-7: x-Koordinate, y-Koordinate des Knotens vom äußeren Polygon

Zeile 8: Anzahl Knoten des 1. inneren Polygons

Zeile 9-11: x-Koordinate, y-Koordinate des Knotens vom 1. inneren Polygon

Zeile 12: Anzahl Knoten des 2. inneren Polygons

Zeile 13-16: x-Koordinate, y-Koordinate des Knotens vom 2. inneren Polygon

## 4. 2 imageCleaner.py

In den Experimenten sowie in der `main.py` werden auch die Methoden von `imageCleaner.py` genutzt, da dieser die Übergabeparameter vom Experiment prüft und ggf. Fehler wirft oder diese selbst behebt.

Außerdem enthält dieses Python-Script auch Methoden im Abschnitt "IMAGE SIZING" für Anpassung und Zuschneiden der Bildgröße sowie Rotieren des Bildes, welche noch in pFlow implementiert werden könnten.

## 4. 3 main.py

Das Python-Script `main.py` ist das Herzstück von pFlowEdge-Detector.

Dieses Script geht wie folgt:

1. Vorverarbeitung erfolgt mit Closing und einem Gaußfilter.
2. Kanten werden mittels Canny-Kantendetektor lokalisiert.
3. Für die Nachverarbeitung wird ein Closing eingesetzt.
4. Ecken werden mittels Harris-Eckendetektor detektiert.
5. Ecken werden bereinigt.
6. Mithilfe des Dijkstra Algorithmus werden die Ecken sinnvoll miteinander verbunden.  
Die Polygone werden erzeugt und abschließend gespeichert.

Nähere Details dazu können aus der unten angegebenen Literatur, aus der Dokumentation im File `main.py` oder aus dem Benutzerhandbuch entnommen werden.

### **Weiterführende Literatur:**

Allgemein:

Wilhelm Burger und Mark James Burge - *Digitale Bildverarbeitung* - 2015<sup>2</sup>

[https://docs.opencv.org/3.4/d7/da8/tutorial\\_table\\_of\\_content\\_imgproc.html](https://docs.opencv.org/3.4/d7/da8/tutorial_table_of_content_imgproc.html)

Edge Preserving Filter:

[https://docs.opencv.org/4.x/df/dac/group\\_photo\\_render.html#gafaae2977597029bc8e35da6e67bd31f7](https://docs.opencv.org/4.x/df/dac/group_photo_render.html#gafaae2977597029bc8e35da6e67bd31f7)

<https://learnopencv.com/non-photorealistic-rendering-using-opencv-python-c/>

Bilateral Filter:

[https://docs.opencv.org/4.x/d4/d86/group\\_imgproc\\_filter.html#ga9d7064d478c95d60003cf839430737ed](https://docs.opencv.org/4.x/d4/d86/group_imgproc_filter.html#ga9d7064d478c95d60003cf839430737ed)

<https://machinelearningknowledge.ai/bilateral-filtering-in-python-opencv-with-cv2-bilateralfilter/>

Für Sobel-Operator:

<https://www.youtube.com/watch?v=uihBwtPIBxM&t=244s>

Literatur für Canny-Kantendetektor:

[https://docs.opencv.org/3.4/da/d5c/tutorial\\_canny\\_detector.html](https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html)

<https://www.youtube.com/watch?v=sRFM5IEqR2w>

Literatur für Harris-Kantendetektor:

[https://docs.opencv.org/3.4/dc/d0d/tutorial\\_py\\_features\\_harris.html](https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html)

Literatur für Dijkstra Algorithmus:

Herbert Süße und Erik Rodner - *Bildverarbeitung und Objekterkennung* - 2014<sup>2</sup>

---

<sup>2</sup> Als E-Book in der HTWG-Bibliothek verfügbar.