TECHNISCHE UNIVERSITÄT
IN DER KULTURHAUPTSTADT EUROPAS
CHEMNITZ

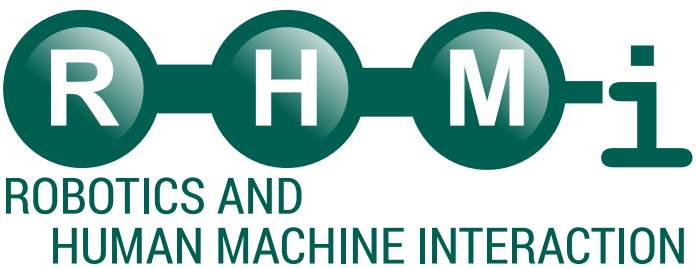
Fakultät für Elektrotechnik und Informationstechnik

Professur für Robotik und Mensch-Technik-Interaktion

– Sommersemester 2023 –

Praktikum: Robotersehen

Versuch 1: Filter



Inhaltsverzeichnis

1 Praktikumsbeschreibung

Innerhalb der verschiedenen Versuche (1 bis 5) wird Stück für Stück eine Bibliothek für die Bildverarbeitung erarbeitet, welche Ihr Verständnis der Bildverarbeitung und deren Methoden stärken soll. Die von Ihnen implementierte/aufgebaute Bibliothek wird dann im sechsten und letzten Versuch verwendet. Dort wird eine Visual Servoing – Robotersehen – Anwendung geschrieben, bei der ein Industrieroboter über die aufgenommenen und verarbeiteten visuellen Informationen gesteuert wird. In Abbildung 1 ist ein beispielhafter Aufbau dargestellt.

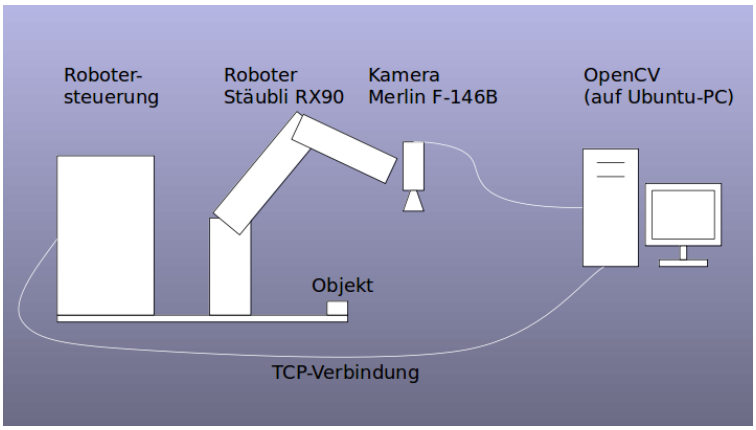


Abbildung 1: Versuchsaufbau Visual Servoing.

Für die Versuche 1 bis 5 steht Ihnen ein Linux-PC und eine USB-Kamera zur Verfügung. Für den Versuch 6 verwenden Sie zusätzlich den iiwa-Roboter der Firma Kuka. Ihr Programme wird in Python geschrieben, wobei Sie die Bibliotheken von OpenCV 3 (package: opencv-python), NumPy (package: python-opencv) und Matplotlib (package: matplotlib) nutzen. Als Programmierumgebung wird die IDE PyCharm (`jetbrains.com/pycharm/`) von JetBrains verwendet. Sie können für die eigene Testprogrammierung auf Ihrem eigenem System auch gern eine andere IDE verwenden, wie Thonny oder IDLE, dies ist aber freiwillig.

Ein wichtiges Ziel des Praktikums neben den Verständnis der Bildverarbeitung ist es, dass Sie Ihre Programmierkenntnisse und -verständnise verbessern und Ihnen das Konzept der objektorientierten Programmierung näher gebracht wird.

Im ersten Versuch soll veranschaulicht werden, welchen Zweck die einzelnen Filterarten haben und bei welchen Anwendungsfällen sie bevorzugt werden sollten. Dazu werden verschiedene Filter auf Bildern angewendet und deren Effekte analysiert. Prinzipiell dienen Filter der Vorverarbeitung und Verbesserung der Qualität von Bilddaten, wie zum Beispiel die Rauschreduktion beziehungsweise der Reduzierung des zufälligen Fehlers. Als Einstieg zum besseren Verständnis der Bilddaten wird von Ihnen als Erstes das Histogramm berechnet und dargestellt. Basierend auf diesem wird dann der erste Filter – der Histogrammausgleich – implementiert. Danach wird mittels der Faltung der Mittelwertfilter erarbeitet und implementiert, welcher dann mit den vorhandenen Gauss- und Bilateralfilter verglichen wird. Als vorletzte Aufgabe soll der Medianfilter realisiert werden. Es wird nach jeder Implementation eines Filters dessen Einflüsse beurteilt und verglichen. Zum Ende des Praktikums werden alle Filter verglichen und die Vor- und Nachteile gemeinsam mit allen Gruppen erörtert.

2 Versuchsvorbereitung

1. Nennen Sie den Speicheraufwand und die Möglichkeiten verschiedener Datenformate für Farbe-, Grau- und Binärbilder.
2. Beschreiben Sie den HSV-Farbraum und benennen Sie dessen Vorteile.
3. Skizzieren Sie ein Histogramm und beschreiben Sie dessen Eigenschaften.
4. Erläutern Sie den vollen Histogrammausgleich. Beschreiben Sie zusätzlich mittels eines Ablaufplanes wie dieser ausführlich implementiert wird.
5. Was ist unter einem verrauschten Bild zu verstehen und nennen Sie die Ursachen dafür? Benennen Sie die verschiedene Rauscharten.
6. Welche linearen und nichtlinearen Filter kennen Sie und wofür werden sie eingesetzt?
7. Erläutern Sie folgende Gleichung:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)F(i-u,j-v) \tag{1}$$

   Welche Operation steckt hinter dieser Gleichung?
8. Führen Sie die Faltung für das Eingangsbild mit dem gegebenen Kernel durch. Die nicht berechenbaren Randbereiche werden mit Nullen gefüllt.

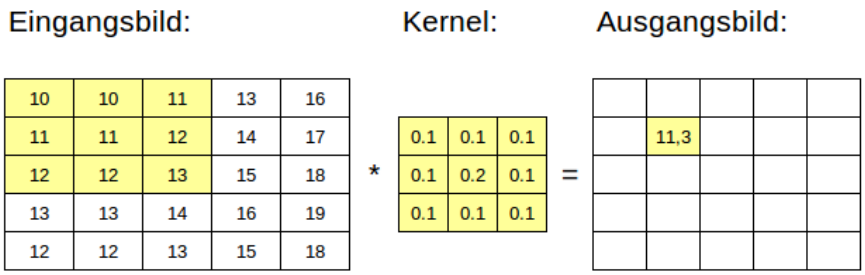


Abbildung 2: Beispiel Faltungsoperation

9. Welche Eigenschaften wird das Ausgangsbild gegenüber dem Eingangsbild aufweisen? Welcher vereinfachte Filter wurde verwendet?
10. Bei einer Faltung besteht immer das Problem mit den Randpixeln. Am Rand stehen nicht genug Nachbarpixel zur Verfügung, um die allgemeine Berechnung durchzuführen. Nennen Sie die verschiedenen Methoden, um mit dem Randbereich umzugehen und beschreiben Sie diese.
11. Erläutern Sie die Filter: Mittelwert, Gauss und Median. Stellen Sie dazu jeweils die Kernel auf (wenn vorhanden) und stellen Sie einen Ablaufplan für jeden Filter auf. Sie benötigen diese für die Implementierung des Mittelwert- und Medianfilters.
12. Sehen Sie sich den vorbereiteten Quellcode in diesem Dokument an und bringen Sie einen Ablaufplan auf Papier für die von Ihnen zu füllenden Stellen zum Kolloquium mit.

3 Versuchsdurchführung

Bei der Versuchsdurchführung sollen Sie die verschiedenen Filter implementieren und deren Ergebnisse diskutieren. **In diesem Versuch ist die Programmierung von fertigen OpenCV-Filterfunktionen nicht erlaubt sind!** Die Filter sollen selbständig implementiert werden, um ein besseres Verständnis von diesen zu erlangen.

In Abbildung 3 ist das UML-Diagramm mit den verschiedenen Filter-Klassen dargestellt. Dabei ist die Basisklasse *FilterBase* eine abstrakte Klasse, da sie hier die Filterfunktion ausschließlich vorgibt, diese aber selbst nicht implementiert. Stattdessen wird eine Programmierung in der Unterklasse verlangt. Dass die Funktion: *filter(image)* und die Klasse: *FilterBase* abstrakt sind, ist über die kursive Schrift in der Abbildung dargestellt. Die einzelnen Filter, welche dann später in der Anwendung verwendet werden, sind alle von der Klasse *FilterBase* abgeleitet. Die Ableitung wird wiederum über den Pfeil im UML-Diagramm dargestellt. Bei den Unterklassen ist die Methode: *filter(image)* implementiert, was über die normale Schrift der Methode dargestellt wird. Dazu besitzen ebenfalls alle Unterklassen das Attribut *kernel_size* der Basisklasse. Dieses Attribut wird dann in der Umsetzung des Filters verwendet. In Abhängigkeit des Filters besitzen einige Filterklassen zusätzliche Attribute, wie im UML-Diagramm ersichtlich ist.

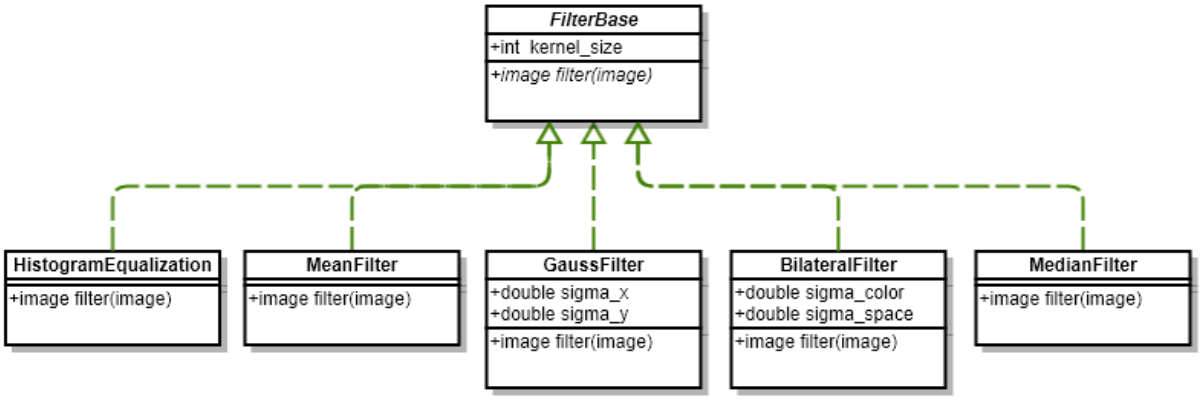


Abbildung 3: UML-Diagramm der Filterklassen.

Zusätzlich zu den Filterklassen wird eine eigene Bildklasse verwendet, die zugehörige UML-Klasse ist in Abbildung 4 visualisiert. Mittels der *Image* Klasse können Sie im Versuch das Bild einladen, anzeigen, manipulieren oder gegebenenfalls diesem ein Rauschen hinzufügen.

Zur Durchführung müssen Sie das Python-Skript in der Datei „mainV1.py“ ausführen. In diesem werden dann die einzelnen Aufgaben schrittweise durchgeführt. Zur Weiterführung des Programms nach der Anzeige eines Bildes, müssen Sie nur eine Taste betätigen.

3.1 Aufgabe 1: Histogramm

Implementieren Sie die Berechnung des Histogramms eines Grauwertbildes in der Klasse *Image*, welche in der Datei „image.py“ implementiert ist. In Abschnitt 4.2 ist die vorbereitete Implementation, der *Image* Klasse dargestellt. Vervollständigen Sie die Funktion *display_histogram* (Zeile 30), der Aufruf ist im Hauptprogramm schon implementiert, Sie

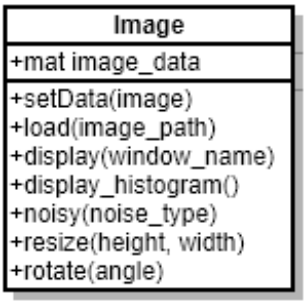


Abbildung 4: UML-Diagramm der *Image* Klasse.

müssen nur das Histogramm berechnen. Des Weiteren werden in diesem Versuch die Bilder als Grauwerte eingelesen, eine Konvertierung ist also nicht notwendig. Zur Weiterführung des Programms nach der Anzeige müssen Sie den Plot einfach schließen.

Lassen Sie sich danach die ermittelten Histogramme für verschiedenen Beispielbilder anzeigen und diskutieren sie, ob die Resultate plausibel sind. Wechseln Sie dazu am Anfang der „mainV1.py“ zwischen den drei Beispielbildern. Testen Sie wie sich das Histogramm ändert, wenn den Bildern ein Rauschen hinzugefügt wird. Dazu könne Sie in der „mainV1.py“ verschiedenes Rauschen hinzufügen, siehe auskommentierte Kommandos.

Welche Eigenschaften der Bilder können jeweils aus dem Histogrammen ermittelt werden? Welche Schlussfolgerungen für die Anwendung von möglichen Bildveränderungen oder Filtern können aufgrund der Ergebnisse der Histogramme gezogen werden?

3.2 Aufgabe 2: Histogrammausgleich

In der zweiten Aufgabe soll der Histogrammausgleich für ein Grauwertbild implementiert werden. Der Quellcode, in welchem Sie die Funktion *filter(image)* implementieren, ist in der Datei „histogram_equalization.py“ (siehe Abschnitt 4.4) geschrieben.

Vergleichen Sie die Histogramme und Bilder vor und nach der Anwendung des Histogrammausgleichs. Welche Unterschiede können Sie erkennen? Diskutieren Sie die Vor- und Nachteile des Histogrammausgleichs.

3.3 Aufgabe 3: Mittelwertfilter

Implementieren Sie den Mittelwertfilter in der Datei „mean_filter.py“ (siehe Abschnitt 4.5). Beachten Sie dabei den Randbereich der Bilder und schreiben eine Methode, um dieses Problem zu lösen. Die verschiedenen Möglichkeiten zur Berechnung der Randpixel wurden hierzu in der Vorlesung vorgestellt. Für den Versuch reicht es aber aus, wenn Sie die Pixel, welche nicht berechnet werden können, mit Nullen füllen.

Wenden Sie den Mittelwertsfilter für die 3 verschiedenen Beispielbilder an und variieren Sie bei der Kernelgröße. Vergleichen Sie ihn zusätzlich mit dem Gauß- und Bilateralfilter, welcher schon implementiert ist (variieren Sie beim Gaußfilter mit den Sigma-Werten).

Welche Unterschiede können Sie erkennen und wann sollten die unterschiedlichen Filterarten verwendet werden?

3.4 Aufgabe 4: Medianfilter

Zuletzt soll der Medianfilter implementiert werden. Ergänzen Sie dazu in der Datei „median_filter.py“ (siehe Abschnitt 4.6) die Funktion: *filter(image)*. Beachten Sie bei der Implementation erneut den Randbereich des Bildes. Bei diesem reicht erneut eine Füllung mit Nullen aus.

Lassen Sie sich das Resultat anzeigen und variieren Sie mit verschiedenen Rauscharten. Diskutieren Sie für welche Rauschart der Medianfilter besonders geeignet ist. Testen Sie zudem verschiedene Filtergrößen und beobachten Sie die Veränderungen.

3.5 Aufgabe 5: Vergleich der Filter

Zuletzt sollen die verschiedene Filter gegeneinander verglichen werden. Am Ende der „mainV1.py“ wird hierfür ein Fenster geöffnet bei dem alle Filterarten nebeneinander dargestellt werden. Testen Sie Resultate der Filter bei verschiedenen Rauscharten und unterschiedlichen Standardabweichungen. Diskutieren Sie die Ergebnisse der unterschiedlichen Filter.

3.6 Plenum

Zum Abschluss des Praktikums werden die, in den Gruppen getroffen, Schlussfolgerungen gemeinsam besprochen. Schreiben Sie sich dazu bitte Ihre Schlussfolgerungen für die einzelnen Aufgaben auf. Welche Filter sind zu bevorzugen oder ist dies Abhängig von der Anwendung? Gibt es Standardwerte, welche für die Größe der Kernel angesetzt werden können?

4 Quellcode des Programms

Das von Ihnen zu ergänzende Programm ist auf mehrere Dateien verteilt. Zum Ersten dient die Datei *mainV1.py* als Hauptprogramm, im gleichen Sinne wie dies die *main.cpp* bei C++ Programmen ist. Diese gibt den Ablauf des kompletten Versuchs vor. Die Zweite Datei ist: *image.py*, welche die Bildklasse des Praktikums ist, diese beinhaltet eine Vielzahl von Methoden für Manipulation und der Anzeige des Bildes. In dieser Datei wird von Ihnen die Histogrammberechnung implementiert. Und zuletzt die *FilterBase* Klasse in *filter.py*, welche das Interface der Filterfunktion für alle abgeleiteten Filterklassen vorgibt. In den abgeleiteten Klassen: *MeanFilter*, *MedianFilter*, *HistogramEqualization* muss von Ihnen der Funktion des Filters implementiert werden.

4.1 Hauptprogramm mainV1.py

```
import numpy as np

from RHMIcv import utils
from RHMIcv.image import Image
from RHMIcv.filter.bilateral_filter import BilateralFilter
from RHMIcv.filter.gauss_filter import GaussFilter
from RHMIcv.filter.histogram_equalization import HistogramEqualization
from RHMIcv.filter.mean_filter import MeanFilter
from RHMIcv.filter.median_filter import MedianFilter

HISTOGRAM = False
HISTOGRAM_EQUALIZATION = False
MEAN_FILTER = True
MEDIAN_FILTER = False

def main():
    # First we will load the images, here some different examples
    img_1 = Image("data/IMG_6960.JPG", 0)
    img_2 = Image("data/Panda_Arm.jpg", 0)
    img_3 = Image("data/mika.png", 0)

    # then we add noise to evaluate the filters and display the result
    img_1.add_noise("gauss")  # or "poison", "s&p"
    img_2.add_noise("poison")  # or "gauss", "s&p"
    img_3.add_noise("s&p")  # or "gauss", "poison"
    img_1.display("original_image")

    if HISTOGRAM is True:
        # first exercise: generate histogram from image (complete function in the file image.py)
        print("Berechnung und Anzeige des Histogramms")
        img_1.display_histogram()

    if HISTOGRAM_EQUALIZATION is True:
        # second exercise: write the histogram equalization (complete function in the file
            histogram_equalization.py)
        print("Berechnung und Anzeige des Histogrammausgleichs")
        histogram_equalization = HistogramEqualization()
        equal_img = histogram_equalization.filter(img_1)
        equal_img.display_histogram()

    if MEAN_FILTER is True:
        # third exercise: write the mean or box filter (complete function in the file mean_filter
            .py)
        print("Berechnung und Anzeige des Mittelwertfilters")
        kernel_size = 3
        mean_filter = MeanFilter(kernel_size)
        mean_img = mean_filter.filter(img_1)
        mean_img.display("mean filter")

        # In addition to compare Gauss and Bilateral filter, functions are already implemented.
        print("Berechnung und Anzeige des Gauss- und Bilateralfilters")
        gauss_filter = GaussFilter(kernel_size)
```

```
        bilateral_filter = BilateralFilter(kernel_size)
        gauss_img = gauss_filter.filter(img_1)
        bil_img = bilateral_filter.filter(img_1)
        utils.display_images(gauss_img.data, bil_img.data, "gaussian / bilateral filter")

    if MEDIAN_FILTER is True:
        # fourth exercise: write the median filter (complete function in the file median_filter.
            py)
        print("Berechnung und Anzeige des Medianfilters")
        kernel_size = 3
        median_filter = MedianFilter(kernel_size)
        median_img = median_filter.filter(img_1)
        median_img.display("median filter")

    if HISTOGRAM and HISTOGRAM_EQUALIZATION and MEAN_FILTER and MEDIAN_FILTER:
        # Display all filtered images together to compare and discuss them.
        print("Vergleich der verschiedenen Filter")
        rows, cols = img_1.data.shape
        scale = 0.8
        img_1.resize(cols * scale, rows * scale)
        mean_img.resize(cols * scale, rows * scale)
        median_img.resize(cols * scale, rows * scale)
        gauss_img.resize(cols * scale, rows * scale)
        bil_img.resize(cols * scale, rows * scale)
        numpy_horizontal_concat = np.concatenate((img_1.data, mean_img.data, median_img.data,
            gauss_img.data,
                                                  bil_img.data), axis=1)
        strung_filters = Image(numpy_horizontal_concat)
        strung_filters.display("images with the filters: raw, mean, median, gauss, bilateral")

if __name__ == '__main__':
    main()
```

4.2 image.py

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

class Image:
    def __init__(self, image=None, variant=-1):
        if image is None:
            self.data = np.zeros((1, 1), int)
        elif isinstance(image, str) and os.path.isfile(image):
            self.load(image, variant)
        else:
            self.data = image
        self.K = np.identity(3)
        self.dist_coeffs = 0

    def set_data(self, image):
        self.data = image

    def load(self, image_path, variant=-1):
        self.data = cv2.imread(image_path, variant)

    def save(self, image_path):
        cv2.imwrite(image_path, self.data)

    def display(self, window_name="default", destroyable=True):
        cv2.imshow(window_name, self.data)
        if destroyable is True:
            cv2.waitKey(0)
            cv2.destroyWindow(window_name)

    def display_histogram(self):
        # create zero array for the histogram
        bins = np.zeros(256, int)

        # for loop, which iterates through the image
        for element in np.nditer(self.data):
```

```
            # has to be implemented from the student
            # start ...
            print("Need to be implemented")

            # ... end

        # plot the resulting histogram normalized
        bins = bins / np.linalg.norm(bins)
        plt.plot(bins, 'b')
        plt.xlim([0, 256])
        plt.show()

    def add_noise(self, noise_typ):
        if noise_typ == "gauss":
            mean = 2.5
            var = 15
            sigma = var ** 0.5

            if self.data.ndim > 2:
                row, col, ch = self.data.shape
                gauss = np.random.normal(mean, sigma, (row, col, ch))
                gauss = gauss.reshape(row, col, ch)
            else:
                row, col = self.data.shape
                gauss = np.random.normal(mean, sigma, (row, col))
                gauss = gauss.reshape(row, col)

            noisy = self.data + gauss.astype(np.uint8)
            self.data = noisy
        elif noise_typ == "s&p":
            s_vs_p = 0.5
            amount = 0.004
            out = np.copy(self.data)
            # Salt mode
            num_salt = np.ceil(amount * self.data.size * s_vs_p)
            coords = [np.random.randint(0, i - 1, int(num_salt)) for i in self.data.shape]
            out[tuple(coords)] = 1
            # Pepper mode
            num_pepper = np.ceil(amount * self.data.size * (1. - s_vs_p))
            coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in self.data.shape]
            out[tuple(coords)] = 255
            self.data = out
        elif noise_typ == "poisson":
            values = len(np.unique(self.data))
            values = 2 ** np.ceil(np.log2(values))
            noisy = np.random.poisson(self.data * values) / float(values)
            self.data = noisy.astype(np.uint8)
        elif noise_typ == "speckle":
            if self.data.ndim > 2:
                row, col, ch = self.data.shape
                gauss = np.random.rand(row, col, ch) * 0.2
                gauss = gauss.reshape(row, col, ch)
            else:
                row, col = self.data.shape
                gauss = np.random.rand(row, col) * 0.2
                gauss = gauss.reshape(row, col)

            noisy = self.data + self.data * gauss
            self.data = noisy.astype(np.uint8)

    def resize(self, height, width, interpolation=cv2.INTER_CUBIC):
        self.data = cv2.resize(self.data, (int(height), int(width)), interpolation)

    def rotate(self, angle):
        rows = self.data.shape[0]
        cols = self.data.shape[1]
        m = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
        self.data = cv2.warpAffine(self.data, m, (cols, rows))

    def get_array(self):
        if len(self.data.shape) > 2:
            image_data = np.array(cv2.cvtColor(self.data, cv2.COLOR_BGR2GRAY))
        else:
```

```
            image_data = np.array(self.data)
        flattened = image_data.flatten()
        return np.array(flattened)

    def undistort(self, K, coeffs):
        # start ...
        print("Need to be implemented")

        # ... end
```

4.3 filter.py

```
import cv2
import numpy as np
import abc
from RHMIcv.image import Image

class FilterBase:
    def __init__(self, kernel_size=3):
        self.kernel_size = kernel_size

    @abc.abstractmethod
    def filter(self, image):
        """Filter the passed image."""
        pass
```

4.4 histogram equalization.py

```
import numpy as np
from RHMIcv.filter.filter import FilterBase
from RHMIcv.image import Image

class HistogramEqualization(FilterBase):

    def filter(self, image):
        image_data = image.data.copy()

        # create zero array for the histogram
        bins = np.zeros(256, int)
        # return cv2.equalizeHist(img)  # test function from OpenCV
        # for loop, which iterates through the image
        for element in np.nditer(image_data):
            # has to be implemented from the student
            # start ...
            print("Need to be implemented")

            # ... end

        # compute the cumulative histogram
        # has to be implemented from the student
        # start ...
        print("Need to be implemented")

        # ... end

        # equalize the image
        rows, cols = image_data.shape
        for element in np.nditer(image_data, op_flags=['readwrite']):
            # has to be implemented from the student
            # start ...
            print("Need to be implemented")

            # ... end

        return Image(image_data)
```

4.5 mean filter.py

```
import abc
import numpy as np
```

```
from RHMIcv.filter.filter import FilterBase
from RHMIcv.image import Image

class MeanFilter(FilterBase):

    def filter(self, image):
        image_data = image.data.copy()
        new_image = image_data

        # has to be implemented from the student
        # start ...
        print("Need to be implemented")

        # ... end

        return Image(new_image)
```

4.6 median filter.py

```
import numpy as np
from RHMIcv.filter.filter import FilterBase
from RHMIcv.image import Image

class MedianFilter(FilterBase):

    def filter(self, image):
        image_data = image.data.copy()
        new_image = image_data

        # has to be implemented from the student
        # start ...
        print("Need to be implemented")

        # ... end

        return Image(new_image)
```