TECHNISCHE UNIVERSITÄT
IN DER KULTURHAUPTSTADT EUROPAS
CHEMNITZ

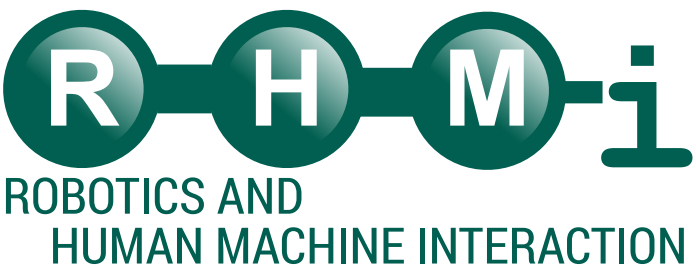
Fakultät für Elektrotechnik und Informationstechnik

Professur für Robotik und Mensch-Technik-Interaktion

– Sommersemester 2023 –

Praktikum: Robotersehen

Versuch 3: Merkmalsextraktion



Inhaltsverzeichnis

1 Praktikumsbeschreibung

Im dritten Versuch soll das Extrahieren von Merkmalen von verschiedenen Bildern durchgeführt werden. Merkmale sind hierbei Bildpunkte, welche wiederzuerkennende Eigenschaften haben. Im besten Falle sollte ein Merkmal immer wieder erkannt werden, unabhängig davon, ob das Bild skaliert und rotiert ist oder eine Blickwinkeländerung besteht. Die Aufgabe der Merkmalsextraktion ist daher die Bereitstellung von geeigneten Merkmalen, diese sind dann die Grundlage für die im nächsten Schritt erfolgende Bildanalyse. Diese beinhaltet häufig eine Klassifikation und Beschreibung der gefundenen Objekte anhand eines bei der Merkmalsextraktion gewonnenen Merkmalsvektors. Neben der Klassifikation von Objekten ist die Extraktion signifikanter Merkmale auch für die Bildregistrierung von Bedeutung. Bei dieser werden zwei oder mehrere Bilder (z.B. medizinische - oder Satellitenbilder), die zu einer Szene gehören, in Übereinstimmung gebracht und zusammengefügt.

In diesem Versuch soll mithilfe verschiedener Algorithmen eine Extraktion von Merkmalen erfolgen. Zuerst sollen einfache Merkmale wie Kanten, Linien und Kreisen erkannt sowie ausgewertet werden. Hierzu wird als 1. der Sobelfilter implementiert, um ein Gradientenbild zu genieren und damit starke Intensitätsänderungen hervorzuheben. Danach wird der Cannyfilter näher untersucht, welcher eine Aneinanderreihung von verschiedenen Filtern und Algorithmen ist, um Kanten in Bildern zu erkennen. Darauf sollen aus diesen Kanten die Linien oder Kreise mit Hilfe der Houghtransformationen extrahiert und evaluiert werden. Darüber hinaus sollen die Ergebnisse von komplexeren Detektoren, wie dem Harris Corner Detektor oder den ORB (Oriented FAST and Rotated BRIEF) Detektor betrachtet werden.

Jeder Algorithmus besitzt eine Vielzahl an Konfigurationsmöglichkeiten. Daher sollen während des Versuchs sogenannte Trackbars implementiert werden. Diese beinhalten Schieberegler und können somit eine interaktive Beeinflussung verschiedener Konfigurationsparameter ermöglichen. Zum Abschluss wird wieder ein Plenum durchgeführt, um die Ergebnisse gegenseitig zu rekapitulieren.

2 Versuchsvorbereitung

1. Erläutern Sie das Ziel der Merkmalsextraktion in der digitalen Bildverarbeitung.

2. Zählen Sie verschiedene Arten von Merkmalen auf (mindestens 3). Wo finden diese jeweils Anwendung?

3. Beschreiben Sie ausführlich die Funktionsweise der folgenden drei Kantenfilter. Geben Sie dazu deren Gleichungen für die Berechnung an und Beschreiben Sie die Bearbeitungsschritte. Darauf basierend nennen Sie die Eigenschaften von diesen Filtern.

   a) Sobel-Filter

   b) Laplace-Filter

   c) Canny-Filter

4. Visualisieren Sie den Ausgang der oben genannten Filter anhand der Funktion aus Abbildung 1, welche die Änderung der Intensität angibt.
   Tipp: Denken Sie an Ihr Schulwissen über Kurven und deren Ableitungen.

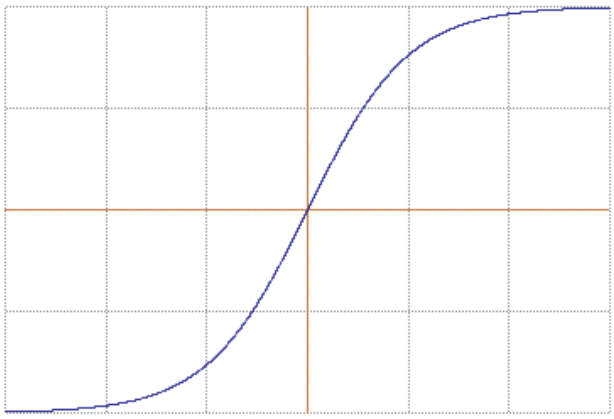


Abbildung 1: Beispielfunktion zur Veranschaulichung des Kantenfilter

5. Erläutern Sie die Wirkungsweise einer Houghtransformation. Wozu dient Sie?

6. Welche Houghtransformation wird ausgeführt, um Linien abzubilden?
   Beschrieben Sie die allgemeine Funktionsweise dieser Transformation und erstellen Sie einen Ablaufplan, um die Houghtransformation selbstständig implementieren zu können.

7. Um verschiedene Merkmale in unterschiedlichen Szenen wiederzufinden, werden sogenannte Deskriptoren verwendet. Welches Problem wird mit diesen gelöst? Nennen Sie Beispiele für 2D Deskriptoren und benennen Sie deren besonderen Eigenschaften.

8. Sehen Sie sich den vorbereiteten Quellcode an und bringen Sie einen Ablaufplan auf Papier für die von Ihnen zu füllenden Stellen zum Kolloquium mit.

3 Versuchsdurchführung

Im Versuch 3 werden Merkmale aus Bildern extrahiert. Hierbei werden Bilddaten verwendet, bei denen mithilfe von Filtern aus dem 1. Versuch das Rauschen entfernt wurde, sowie eine Entzerrung der Bilddaten über ein Kalibrierung wie in Versuch 2 geschehen ist. Diese beiden Schritte sind ansonsten immer notwendig, um eine bestmögliche Merkmalsextrahierung zu ermöglichen. Es bauen des Weiteren viele der zu untersuchenden Algorithmen auf den Implementierungen aus Versuch 1 und 2 auf. Um zwischen den einzelnen Aufgaben im Hauptprogramm zu wechseln, ändern Sie einfach die Flags: *SOBEL*, *CANNY*, usw. auf *True* beziehungsweise *False*.

3.1 Sobelfilter

Implementieren Sie die Methoden für den Sobelfilter in der Datei „sobel_filter.py“, dies sind die Funktionen: *filter(image)* und *filter_both_sides(image)*. Beachtung Sie, dass für den einfachen Sobelfilter der Richtungsparameter per Set-Methode definiert wird. Hierbei können eigene Implementierungen oder die vorgefertigten OpenCV Befehle genutzt werden. Beachten Sie dabei die besonderen Eigenschaften des Sobelfilters.

Testen Sie darauf die verschiedenen Richtungen des Sobelfilers an den Beispielbildern im Hauptprogramm. Wenden Sie den Filter so an, dass für das Bild „sobel_test.png“ ausschließlich der äußere Ring des Kreises sichtbar ist.

Diskutieren Sie in der Gruppe. Welche Einfluss haben die beiden Richtungsparameter zueinander? Ist die Nutzung von nur einem Richtungsparameter sinnvoll? Welchen Einfluss hat der Datentyp der Bildmatrix?

3.2 Cannyfilter

Der sogenannte Cannyfilter gilt als Erweiterung des Sobelfilters und filtert zusammenhängende Linien aus einem Grauwertbild. Durch die Kombination von verschiedenen Filtern und Methoden, führt er zu wesentlich besseren Resultaten.

3.2.1 Implementierung

Implementieren Sie die Methode: *filter(image)* für den Cannyfilter in der Datei „canny_filter.py“. Hierbei können eigene Implementierungen sowie vorgefertigte OpenCV Befehle genutzt werden. Beachten Sie die verschiedenen einstellbaren Parameter des Filters (Kernelgröße und Schwellwert), welche wieder mittels Set-Funktion implementiert werden.

Wenden Sie den Filter auf das Bild „sobel_test.png“ an.

3.2.2 Trackbar

Oftmals wird das Ergebnis von Algorithmen durch diverse Parameter beeinflusst. Die Bestimmung der besten Werte hängt von verschiedenen Einflüssen ab und kann somit zu

einem großen Zeitaufwand führen. Daher empfiehlt sich eine flexible Einstellbarkeit, wie sie zum Beispiel durch die „Trackbars“ aus OpenCV gegeben ist, zu nutzen.

Testen Sie die Parameter des Cannyfilter Verwenden Sie das Bild „data/IMG_6960.JPG“, um die Einflüsse der Parameter zu testen und zu diskutieren.

3.3 Hough lines

Um Merkmale, wie z.B. Linien, aus Bildern zu extrahieren kann der sogenannte Houghraum genutzt werden. Implementieren Sie zuerst mithilfe von OpenCV Funktionen eine Liniendetektion und visualisieren Sie die größte ermittelte Linie eines Bildes. Nutzen Sie dazu die Basisklasse aus „feature/feature.py“ und die vorgefertigte Klasse „HoughLinesDetector“ aus der Datei „hough_lines.py“, in welcher Sie die Methode *hough_lines_opencv* vervollständigen.

Im zweiten Schritt sollen Sie anschließend eine vergleichbare Funktion implementieren, die vollständig auf OpenCV Funktionen verzichtet. Vervollständigen Sie dazu die Methode *hough_lines_self*.

Welche Vor- und Nachteile fallen Ihnen bei einem Vergleich der Implementierung und deren Ergebnisse auf? Kann der Algorithmus ohne eine vorhergehende Kantenerkennung verwendet werden?

3.4 Hough circles

Weitere interessante Merkmale, die aus Bildern ermittelt werden können, sind zum Beispiel Kreise. Nutzen Sie die Erkenntnisse aus den vorigen Aufgaben, um die Kreise in der Datei „circles.png“ zu ermitteln. Untersuchen Sie dabei den Einfluss der verschiedenen Parameter, der OpenCV Funktion „HoughCircles“. Vervollständigen Sie dazu die Methode *hough_cirles* in der Datei „hough_cirles.py“.

Ermitteln Sie passende Parameter, um die Kreise im Bild automatisch zählen zu können. Was fällt Ihnen dabei auf?

3.5 Harris Detektor

Merkmalsdeskriptoren können an allen Positionen im Bild ermittelt werden, um signifikante Punkte herauszufiltern kann zum Beispiel der Harris Corner Detektor angewendet werden. Implementieren Sie diesen mithilfe von OpenCV Funktionen in der Datei „harris_feature.py“. Beachten Sie dabei, dass nur Grauwertbilder verarbeitet werden können. Markieren Sie die Ergebnisse der wichtigsten Punkte farblich im Originalbild (minimaler Schwellwert z.B. 0,02).

Wie beeinflusst der Schwellwert die Merkmalserkennung und wo taucht er in der allgemeinen Berechnung des Harris Detektor auf?

3.6 ORB Merkmalsvektor

3.6.1 Grundlagen

Die Ermittlung der ORB (**O**riented FAST and **R**otated **B**RIEF) Merkmalsvektoren ist eine Kombination aus verschiedenen bereits bestehenden Verfahren und gilt als Alternative zu SURF- oder SIFT-Vektoren. ORB Vektoren weisen eine ähnliche Genauigkeit und eine geringere Berechnungszeit auf. Begonnen wird mit der Ermittlung von signifikanten Punkten. Diese werden mit dem FAST-Verfahren ermittelt, wobei zu jedem Punkt im Bild ein Intensitsätsvektor gebildet wird, der sich aus den jeweiligen umliegenden Punkten ermittelt. Diese Vektoren werden mit Hilfe des Harris Corner Detektors verfeinert und nach Signifikanz geordnet. Die Signifikantesten Punkte werden herausgefiltert und als Intensitätsmittelpunkt bezeichnet. An diesen Punkten werden mit Hilfe des BRIEF-Verfahren Binärvektoren bestimmt. Dabei werden ausgehend von jedem Punkt in einem definierten Radius Imagepatches (Bildausschnitte) erstellt. Aus diesen werden zufällige Punktpaare herausgefiltert und deren Helligkeitsdifferenzen ermittelt. Wenn der erste Punkt des Punktpaares einen höheren Intensitätswert als der zweite Punkt besitzt, so wird dem Vektor eine 1 hinzugefügt, andernfalls eine 0. Dies wird ca. 128 mal wiederholt.

3.6.2 Verwendung

In OpenCV verfügt über eine Vielzahl an Algorithmen für die Ermittlung von Merkmalsvektoren. Implementieren Sie als Beispiel die Detektion der ORB Vektoren in der Datei „orb_feature.py“ mithilfe der Methoden: *calc_keypoints* und *draw_points*. Der Algorithmus kann wiederum durch verschiedene Parameter beeinflusst werden. Implementieren Sie eine Möglichkeit um den Parameter „nfeatures“ zu verändern und visualisieren zwei verschiedene Ergebnisse.

Testen Sie darüber hinaus die Ermittlung der Vektoren anhand verschiedener Rotationen und Größen des Originalbildes. Was fällt Ihnen bezüglich der Positionen der Merkmale auf?

3.7 Anwendungsbeispiel zur Objekterkennung

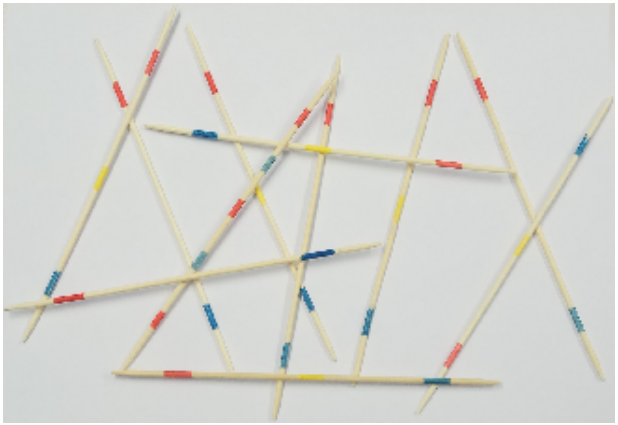
Zum Abschluss des Versuches sollen die Erkenntnisse des 1. und 3. Versuches zur Objekterkennung genutzt werden. Führen Sie dazu die einzelnen Funktionen aus den bisherigen Versuchen zu einem Programm zusammen, das in der Lage ist automatisiert die Anzahl an Münzen aus Abbildung 2a und die Anzahl an Mikadostäbchen aus Abbildung 2b zu detektieren.

3.8 Plenum

Diskutieren Sie gemeinsam mit allen Gruppen über das Praktikum. Dabei sollen folgende Fragen als Basis dienen:

- Ist eine Merkmalsextrahierung ohne Deskriptoren sinnvoll?
- Welche Anwendungen können Sie sich für die Merkmalsextrahierung vorstellen?

(a) Münzen



(b) Mikado

- Warum sollten die Algorithmen invariant zu Skalierung, Rotation und Blickwinkel sein?

4 Quellcode des Programms

Das von Ihnen zu ergänzende Programm ist auf mehrere Dateien verteilt. (Siehe andere Versuche)

4.1 Hauptprogramm mainV3.py

```
import numpy as np
import cv2

from RHMIcv import utils
from RHMIcv.image import Image

from RHMIcv.filter.sobel_filter import SobelFilter
from RHMIcv.filter.canny_filter import CannyFilter
from RHMIcv.trackbars.trackbar_canny import CannyTrackbar

from RHMIcv.feature.hough_lines import HoughLinesDetector
from RHMIcv.feature.hough_circles import HoughCirclesDetector
from RHMIcv.trackbars.trackbar_hough_circles import HoughTrackbar

from RHMIcv.feature.harris_feature import HarrisFeature
from RHMIcv.feature.orb_feature import ORBFeature

SOBEL = False
CANNY = False
HOUGH_CV = False
HOUGH_SELF = False
HOUGH_CIRCLE_DEMO = True
HARRIS = True
ORB = True
APPLICATION_EXAMPLE = False

def main():
    # load and show the image
    img_1 = Image("data/IMG_6960.JPG", cv2.IMREAD_GRAYSCALE)
    img_sobel = Image("data/sobel_test.png", cv2.IMREAD_GRAYSCALE)
    img_muenzen = Image("data/muenzen.png")
    img_mika = Image("data/mika.png", cv2.IMREAD_GRAYSCALE)
    img_1.display("original image")

    # first exercise: the sobel filter
    if SOBEL is True:
        # implement sobel filter for different directions
        sobel_filter = SobelFilter(5)  # initialize the filter

        sobel_filter.set_directions(1, 0)  # x, y
        sobel_x_img = sobel_filter.filter(img_1)

        sobel_filter.set_directions(0, 1)
        sobel_y_img = sobel_filter.filter(img_1)

        utils.display_images(sobel_x_img.data, sobel_y_img.data, "x / y")

        sobel_both_img = sobel_filter.filter_both_sides(img_sobel)
        sobel_both_img.display("sobel both")

    # second exercise: canny filter with trackbar
    if CANNY is True:
        canny_filter = CannyFilter()  # initialize the filter
        canny_img = canny_filter.filter(img_1)
        canny_img.display("canny")

        trackbar = CannyTrackbar(canny_filter)  # initialize the filter
        trackbar.set_image(img_1)  # change image to test canny filter
        trackbar.trackbar_setup("kernel", trackbar.on_trackbar_kernel)
        trackbar.trackbar_setup("lower", trackbar.on_trackbar_min)
        trackbar.trackbar_setup("upper", trackbar.on_trackbar_max)
        cv2.waitKey(0)
```

```
    # third exercise a): Hough Lines Detector with OpenCV
    if HOUGH_CV is True:
        canny_filter = CannyFilter()
        canny_filter.set_parameter(400, 600, 3)
        filtered_image = canny_filter.filter(img_1)
        filtered_image.display("Kanten vom Canny-Filter")

        # compute the lines and display them
        hough_lines = HoughLinesDetector()
        hough_lines.set_image(filtered_image)
        lines_image = hough_lines.hough_lines_opencv(150)
        lines_image.display("Hough Lines")

    # third exercise b): Hough Lines Detector by yourself
    if HOUGH_SELF is True:
        canny_filter = CannyFilter()
        canny_filter.set_parameter(400, 600, 3)
        filtered_image = canny_filter.filter(img_1)
        filtered_image.display("Kanten vom Canny-Filter")

        # compute the lines and display them
        hough_lines = HoughLinesDetector()
        hough_lines.set_image(filtered_image)
        lines_image = hough_lines.hough_lines_self()
        lines_image.display("Hough Lines Self")

    # fourth exercise: hough circle detection with trackbar
    if HOUGH_CIRCLE_DEMO is True:
        hough_circles = HoughCirclesDetector()
        trackbar = HoughTrackbar(hough_circles)

        trackbar.set_image(img_muenzen)
        trackbar.trackbar_setup("threshold_1", trackbar.on_trackbar_threshold_1)
        trackbar.trackbar_setup("threshold_2", trackbar.on_trackbar_threshold_2)
        trackbar.trackbar_setup("min_rad", trackbar.on_trackbar_min_radius)
        trackbar.trackbar_setup("max_rad", trackbar.on_trackbar_max_radius)
        cv2.waitKey(0)

    if HARRIS is True:
        harris_detector = HarrisFeature()
        harris_detector.set_image(img_1)
        harris_corners = harris_detector.calc_keypoints()
        harris_corners.display("Harris keypoints")
        harris_corners = harris_detector.refined_keypoints()
        harris_corners.display("Harris keypoints refined")

    if ORB is True:
        orb_detector = ORBFeature()
        orb_detector.set_image(img_1)
        orb_detector.set_features(30)
        orb_detector.calc_keypoints()
        image_with_orb_points = orb_detector.draw_points(draw_orientation=True)

        rot_image = img_1
        rot_image.rotate(45)
        rows, cols = rot_image.data.shape
        rot_image.resize(cols * 1.2, rows)
        orb_detector.set_image(rot_image)
        orb_detector.calc_keypoints()
        image_with_orb_points_2 = orb_detector.draw_points(draw_orientation=True)
        utils.display_images(image_with_orb_points.data, image_with_orb_points_2.data, "orb")

    if APPLICATION_EXAMPLE is True:
        # start ...
        print("Need to be implemented")

        # ... end

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

4.2 Filter

4.2.1 Sobel Filter

```
import cv2
import numpy as np
from RHMIcv.filter.filter import FilterBase
from RHMIcv.image import Image

class SobelFilter(FilterBase):
    def __init__(self, kernel_size=3):
        FilterBase.__init__(self, kernel_size)
        self.direction_x = 1
        self.direction_y = 0

    def set_directions(self, x, y):
        self.direction_x = x
        self.direction_y = y

    def filter(self, image, image_type=cv2.CV_64F):
        img = image.data.copy()
        # start ...
        print("Need to be implemented")

        # ... end
        return Image(output_image)

    def filter_both_sides(self, image, image_type=cv2.CV_64F):
        # start ...
        print("Need to be implemented")

        # care of absolute values

        # ... end

        return Image(image_x + image_y)
```

4.2.2 Canny Filter

```
import cv2

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

class CannyFilter(FilterBase):
    def __init__(self):
        FilterBase.__init__(self)
        self.lower_bound = 0
        self.upper_bound = 0

    def set_parameter(self, lower_bound, upper_bound, kernel_size):
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.kernel_size = kernel_size

    def filter(self, image):
        if image.data.size is 0:
            return image

        image_data = image.data.copy()
        # start ...
        print("Need to be implemented")

        # ... end

        return Image(output_image)
```

4.3 Feature

4.3.1 Feature

```
from RHMIcv.image import Image
```

```
class FeatureBase:
    def __init__(self):
        self.orig_img = Image()

    def set_image(self, input_image):
        self.orig_img = input_image
```

4.3.2 Hough Lines

```
import cv2
import math
import numpy as np

from RHMIcv.image import Image
from RHMIcv.feature.feature import FeatureBase

class HoughLinesDetector(FeatureBase):
    def __init__(self):
        FeatureBase.__init__(self)

    def hough_lines_opencv(self, threshold):
        if len(self.orig_img.data.shape) > 2:
            self.orig_img.data = cv2.cvtColor(self.orig_img.data, cv2.COLOR_BGR2GRAY)

        print("detecting␣lines␣by␣opencv")
        # start ...
        print("Need␣to␣be␣implemented")

        # ... end

        cdst = cv2.cvtColor(self.orig_img.data, cv2.COLOR_GRAY2BGR)
        if lines is None:
            print("No␣lines␣detected!")
            return Image(cdst)
        for i in range(0, len(lines)):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            pt1, pt2 = self.angles_to_points(rho, theta, 1000)
            cv2.line(cdst, pt1, pt2, (0, 0, 255), 3, cv2.LINE_AA)
        return Image(cdst)

    def hough_lines_self(self):
        if len(self.orig_img.data.shape) > 2:
            self.orig_img.data = cv2.cvtColor(self.orig_img.data, cv2.COLOR_BGR2GRAY)

        print("detecting␣lines␣by␣my␣self")
        accumulator, thetas, rhos = self.hough_lines_calc(self.orig_img.data)
        # start ...
        print("Need␣to␣be␣implemented")

        # ... end
        c_dst = cv2.cvtColor(self.orig_img.data, cv2.COLOR_GRAY2BGR)
        cv2.line(c_dst, pt1, pt2, (0, 0, 255), 3, cv2.LINE_AA)
        return Image(c_dst)

    @staticmethod
    def hough_lines_calc(img):
        # start ...
        print("Need␣to␣be␣implemented")

        # Rho and Theta ranges

        # Cache some reusable values

        # Hough accumulator array of theta vs rho

        # Vote in the hough accumulator

        # ... end
        return accumulator, thetas, rhos

    @staticmethod
```

```
    def angles_to_points(rho, theta, length):
        # start ...
        print("Need to be implemented")

        # ... end
        return pt1, pt2
```

4.3.3 Hough Circles

```
import cv2

import numpy as np

from RHMIcv.image import Image
from RHMIcv.feature.feature import FeatureBase

class HoughCirclesDetector(FeatureBase):
    def __init__(self):
        FeatureBase.__init__(self)
        self.threshold_1 = 50
        self.threshold_2 = 50
        self.min_rad = 100
        self.max_rad = 500

    def set_parameter(self, par1=50, par2=50, min_radius=100, max_radius=500):
        self.threshold_1 = par1
        self.threshold_2 = par2
        self.min_rad = min_radius
        self.max_rad = max_radius

    def hough_circles(self):
        if len(self.orig_img.data.shape) > 2:
            self.orig_img.data = cv2.cvtColor(self.orig_img.data, cv2.COLOR_BGR2GRAY)

        # start ...
        print("Need to be implemented")

        # ... end

        return circles

    def draw_circles(self, circles):
        if circles is not None:
            circles = np.uint16(np.around(circles))
            image_to_draw = self.orig_img.data.copy()
            for i in circles[0, :]:
                # draw the outer circle
                cv2.circle(image_to_draw, (i[0], i[1]), i[2], (0, 255, 0), 2)
                # draw the center of the circle
                cv2.circle(image_to_draw, (i[0], i[1]), 2, (0, 0, 255), 3)
            return Image(image_to_draw)
        else:
            return Image(self.orig_img.data.copy())

    @staticmethod
    def count_circles(circles):
        if circles is not None:
            number = len(circles[0, :])
            return number
        else:
            return 0
```

4.3.4 Harris

```
import cv2

import numpy as np

from RHMIcv.image import Image
from RHMIcv.feature.feature import FeatureBase

class HarrisFeature(FeatureBase):
```

```
    def __init__(self):
        FeatureBase.__init__(self)

    def calc_keypoints(self, block_size=2, aperture_size=3, k=0.04):
        # start ...
        print("Need to be implemented")

        # ... end

    def refined_keypoints(self):
        # start ...
        print("Need to be implemented")

        # ... end
```

4.3.5 ORB

```
import cv2

from RHMIcv.image import Image
from RHMIcv.feature.feature import FeatureBase

class ORBFeature(FeatureBase):
    def __init__(self):
        FeatureBase.__init__(self)
        self.orb = cv2.ORB_create()
        self.keypoints = []

    def set_features(self, num_features):
        # start ...
        print("Need to be implemented")

        # ... end

    def calc_keypoints(self):
        # start ...
        print("Need to be implemented")

        # ... end

    def draw_points(self, draw_orientation=False):
        if draw_orientation is False:
            result_image = cv2.drawKeypoints(self.orig_img.data, self.keypoints, None)
        else:
            result_image = cv2.drawKeypoints(self.orig_img.data,
                                             self.keypoints, None, flags=cv2.
                                                 DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

        return Image(result_image)
```