TECHNISCHE UNIVERSITÄT
IN DER KULTURHAUPTSTADT EUROPAS
CHEMNITZ

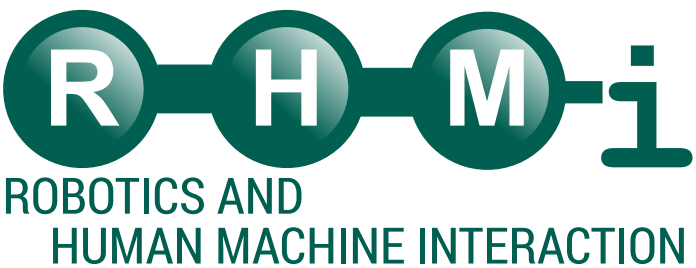
Fakultät für Elektrotechnik und Informationstechnik

Professur für Robotik und Mensch-Technik-Interaktion

– Sommersemester 2023 –

Praktikum: Robotersehen

Versuch 2: Kamerakalibrierung



Inhaltsverzeichnis

1 Versuchsbeschreibung

Im zweiten Versuch wird die Kalibrierung von Kameras durchgeführt. Die Kamerakalibrierung beinhaltet die Ermittlung der intrinsischen und extrinsischen Kameraparameter. Mit den intrinsischen Parametern werden ebenfalls die Verzerrungskoeffizienten bestimmt, mit denen die Auswirkungen von geometrischen Verzerrungen kompensiert werden. Diese Verzerrungen beziehungsweise die Verzeichnungen entstehen durch geometrische Abbildungsfehler der optischen Systeme.

Bei der Kamerakalibrierung soll veranschaulicht werden, dass die Schätzung der Kameraparameter ein Optimierungsproblem darstellt, welches von vielen Koeffizienten abhängig ist. Prinzipiell dienen die intrinsischen Kameraparameter zur Beschreibung von konstruktiven, optischen und geometrischen Eigenschaften der Kamera. Grundsätzlich sind die Parameter abhängig von der Bauweise und Anordnung der Linse sowie dem Sensor. Die extrinsischen Kameraparameter beschreiben im Gegensatz dazu die Position und Orientierung des Kamerakoordinatensystems relativ zu einem gewählten Weltkoordinatensystem. Als Grundlage für die Kompensation der Verzerrungen zu kompensieren, müssen die intrinsischen Kameraparameter als Kameramatrix hinreichend parametrisiert werden.

Der erste Teil des Praktikums ist die Aufnahme von Kalibriermustern. Hierzu nehmen Sie Serien verschiedener perspektivischer Bilder von bekannten Objekten auf. Ein Kalibrierkörper bzw. ein Kalibriermuster wird mit primitiver Geometrie und Struktur mit der verwendeten Kamera mit identischen Einstellungen (Fokus, Vergrößerung und Tiefe) mehrmals abgebildet. Grundlegend besteht das Kalibriermuster aus einer periodischen Anordnung von schwarz und weißen Rechtecken, welche eine definierte Entfernung haben. Als Tool wird die OpenCV Bibliothek verwendet, über welche die Muster erkannt werden und danach darauf basierend die Kalibrierungen geschätzt wird. Basierend auf den perspektivischen Bildern des Kalibriermusters führen Sie die intrinsische Kamerakalibrierung durch und schätzen dabei die Kameramatrix für die verwendete Kamera. In der vorletzten Aufgabe versuchen Sie die radiale Verzerrung entfernen und als letzte Programmieraufgabe schätzen Sie die Pose zwischen Kamera und einem Kalibriermuster. Zum Abschluss wird wieder ein Plenum mit allen Gruppen durchgeführt und es werden Resultate sowie Erkenntnisse besprochen.

2 Versuchsvorbereitung

1. Skizzieren Sie die Lochkameramodelle in den Varianten, bei der das Kamerakoordinatensystem vor bzw. nach der Bildebene positioniert ist.

2. Beschreiben Sie anhand einer Skizze des Linsenmodelles die Begriffe Welt-, Kamera- und Bildkoordinatensystem, optische Achse, Bildebene, Bildhauptpunkt, Brennweite und Objekttiefe.

3. Zählen Sie die extrinsischen und intrinsischen Kameraparameter auf und erläutern Sie ihre Bedeutung. Schreiben Sie dazu die Kameramatrix $\boldsymbol{K}$ auf und beschreiben Sie die Projektionsgleichung.

4. Welcher mathematischer Zusammenhang liegt bei der perspektivischen Transformation zwischen den Welt- und Bildkoordinaten vor?

5. Aus welchen Gründen ist es zumeist notwendig, ein Kamerasystem vor seiner Verwendung zu kalibrieren?

6. Wieso wird die Fokuslänge bzw. Brennweite in der Kameramatrix als 2D-Koordinaten f_x und f_y angeben?

7. Nennen Sie zwei wesentliche Arten geometrischer Verzerrungen und beschreiben Sie ihre physikalischen Ursachen.

8. Erläutern Sie einen allgemeinen mathematischen Algorithmus zur Kompensierung der Verzerrungen.

9. Nennen Sie Anwendungen bei denen eine extrinsische Kalibrierung notwendig ist.

3 Versuchsdurchführung

Bei der Versuchsdurchführung sollen Sie mit der Ihnen zur Verfügung gestellten USB-Kamera und Kalibriermuster verschiedene perspektivische Bilder aufnehmen und als Bildserie lokal speichern. Anschließend führen Sie die Kamerakalibrierung durch und berechnen die intrinsischen Kameraparameter. Abschließend kompensieren Sie mit einem geeigneten mathematischen Polynom die entstandene Bildverzerrung auf Basis der bestimmten Kameramatrix und der Verzerrungskoeffizienten.

3.1 Aufgabe 1: Bildaufnahme vom Kalibriermuster

Schließen Sie die am Arbeitsplatz liegende Webcam an einen freien USB-Port an.

Setzen Sie das Flag *IMAGE_CAPTURE* auf true und führen Sie dann die Datei „mainV2.py“ aus und warten Sie bis der Livestream der Webcam angezeigt wird. Durch den Tastendruck „s“ können Sie Bilder in den Ordner „data/capture“ speichern und mit einem Tastendruck von „q“ können Sie den Videostream beenden.

Speichern Sie mindestens 12 perspektivische Bilder mit folgenden Eigenschaften:

- Zwei unterschiedliche Entfernungen vom Kalibriermuster parallel zur Kameraebene (ohne Hintergrund und mit Hintergrund)
- Vier perspektivische Bilder (links, rechts, nach vorne, nach hinten gekippt) je Entfernung
- Zwei Bilder beliebig gedreht je Entfernung

Hinweis: Es muss immer das vollständige Kalibriermuster sichtbar sein.

Prüfen Sie die Bildserie auf Vollständigkeit und schauen Sie sich die Linien der abgebildeten Rechtsecke des Kalibriermusters genau an. Was stellen Sie fest?

3.2 Aufgabe 2: Bestimmung der Kalibrierpunkte und intrinsische Kalibrierung

Im ersten Schritt werden die abgespeicherten Bilder aus dem Verzeichnis „data_captured“ eingeladen. Danach müssen die Eigenschaften des Targets (Kalibriermuster) definiert werden, das heißt, wie viele Eckpunkte das Muster hat und in welchem Abstand diese zueinander liegen.

Sobald dies geschehen ist, müssen die Punkte des Kalibriermusters ermittelt werden. Vervollständigen Sie dazu in der Datei „RHMIcv/calib/calib.py“ die Funktion „detect_chessboard“ zur Erkennung der Punkte des Kalibriermusters. Dabei sollen die Bildpunkte des Musters mit Subpixelgenauigkeit ermittelt werden. Das heißt, die Positionsangabe der Eckpunkte soll keine ganze Zahl sein und auf dem Pixel liegen. Stattdessen soll die genaue Lage per float-Wert geschätzt werden, welche natürlich zwischen Pixeln liegen kann.

Wenn diese fertig ist, können die Punkte der Muster von allen aufgenommenen Bilder ermittelt werden. Mittels dieser wird dann die Kamera selber kalibriert. Vervollständigen Sie dazu die Funktion „calibrate_intrinsic“ in der Datei „RHMIcv/calib/calib.py“, sodass die Kalibriermatrix $\boldsymbol{K}$ und die Verzeichnungskoeffizienten zurückgegeben werden. Berechnen Sie ebenfalls den Reprojektionsfehler für eine geglückte Kalibrierung und lassen Sie sich diesen anzeigen. Im letzten Schritt wird die ermittelte Kalibrierung für die nächsten Aufgaben gespeichert.

Was stellen Sie fest, wenn Sie alle detektierten Kalibrierpunkte betrachten? Was bedeutet der minimale Reprojektionsfehler und wieso ist er wichtig bei der Kamerakalibrierung? Diskutieren Sie Ihren bestimmten minimalen Reprojektionsfehler RMS.

3.3 Aufgabe 3: Kompensation der Bildverzerrung

Nach der geglückten Kalibrierung wollen wir die Bilder entzerren und damit die Qualität der Verzeichnungskoeffizienten (und indirekt der Kalibrierung) kontrollieren. Als Erstes werden hierzu alle Bilder im Verzeichnis „data_captured“ und die Kalibrierung geladen. Programmieren Sie dazu in der Datei „RHMIcv/calib/calib.py“ die Funktion „undistort“. Sobald dies geschehen ist, können alle Bilder entzerrt und angezeigt werden.

Welche Unterschiede können Sie bei den entzerrten Bildern erkennen? Inwiefern sollte das entzerrte Bild beschnitten werden?

3.4 Aufgabe 4: Extrinsische Kalibrierung

In der letzten Aufgabe soll eine extrinsische Kalibrierung vollzogen werden. Dabei wollen wir die Transformation zwischen Kamerakoordinatensystem und Schachbrett ermitteln. Als Erstes wird hierzu ein Beispielbild im Verzeichnis „data_captured“ und die Kalibrierung geladen. Danach müssen wieder die Parameter des Schachbrettes festgelegt werden. Zur Lösung programmieren Sie die Funktion „calibrate_extrinsic“ in der Datei „RHMIcv/calib/calib.py“ und geben Sie die Rotationsmatrix $\boldsymbol{R}$ und den Translationsvektor $\boldsymbol{t}$ zurück.

Diskutieren Sie das Ergebnis und testen Sie verschiedene Aufnahmen.

3.5 Plenum

Zum Ende des Praktikums werden erneut die Erkenntnisse und Schlussfolgerungen gemeinsam mit allen Gruppen besprochen. Dazu können folgende Fragen als Basis dienen:

- Wofür ist eine Kalibrierung notwendig?
- Welche Genauigkeiten sollten bei dieser erreicht werden?
- Sind gute Linsen notwendig oder können Verzeichnungen gut über eine Entzerrung vermieden werden?
- Welcher Rechenaufwand ist für eine Entzerrung überhaupt notwendig?

4 Quellcode des Programms

Das von Ihnen zu ergänzende Programm ist auf mehrere Dateien verteilt. Zum Ersten dient die Datei *mainV2.py* als Hauptprogramm. 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 Anzeige des Bildes. In dieser Datei wird von Ihnen die Kompensation der Verzerrung implementiert. Und zuletzt die *calib.py*, in welcher die Kalibrierungen und Schachbretterkennung programmiert werden.

4.1 Hauptprogramm mainV2.py

```
import numpy as np
import cv2

from RHMIcv import utils
from RHMIcv.image import Image
from RHMIcv.calib import calib

IMAGE_CAPTURE = False
CALIBRATION = False
DISTORTION_CORRECTION = False
EXTRINSIC_CALIBRATION = True

def main():
    # define chessboard properties
    num_x = 10  # number of corners in x direction (vertical)
    num_y = 7  # number of corners in y direction (horizontal)
    square_size = 20  # size in mm of a square from the calibration target

    if IMAGE_CAPTURE is True:
        image = Image()
        cap = cv2.VideoCapture(0)  # Define general video driver

        count = 0
        while True:
            ret, frame = cap.read()  # Capture frame-by-frame

            image.data = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            image.display("live stream", False)
            key = cv2.waitKey(1)
            if key == ord('q'):
                break
            elif key == ord('s'):
                image_file = "image" + str(count) + ".png"
                print("Write image: " + image_file)
                image.save("data\\capture\\" + image_file)
                count += 1

        # When everything done, release the capture
        cap.release()
        cv2.destroyAllWindows()

    if CALIBRATION is True:
        calib_images = utils.read_captured_images()  # read all images inside the folder "data/
            capture"

        # detect all chessboards and get the object and image points
        obj_pts, img_pts = calib.detect_chessboard(calib_images, num_x, num_y, square_size,
            show_images=True)

        # calibrate and get the camera matrix and the distortion parameters
        K, dist_coeffs = calib.calibrate_intrinsic(obj_pts, img_pts, calib_images[0])
        # save and print the calibration
        cv_file = cv2.FileStorage("calib.xml", cv2.FILE_STORAGE_WRITE)
        cv_file.write("K", K)
        cv_file.write("dist_coeffs", dist_coeffs)
```

```
        cv_file.release()
        print("K: ", K)
        print("distortion coefficients: ", dist_coeffs)

    if DISTORTION_CORRECTION is True:
        calib_images = utils.read_captured_images()  # read all images inside the folder "data/
            capture"
        # read the calibration
        cv_file = cv2.FileStorage("calib.xml", cv2.FILE_STORAGE_READ)
        K = cv_file.getNode("K").mat()
        dist_coeffs = cv_file.getNode("dist_coeffs").mat()

        # test the calibration with the input images, discuss the changes
        for img in calib_images:
            img.undistort(K, dist_coeffs)
            img.display("undistorted")
            cv2.waitKey(0)

    if EXTRINSIC_CALIBRATION is True:
        # read the calibration
        cv_file = cv2.FileStorage("calib.xml", cv2.FILE_STORAGE_READ)
        K = cv_file.getNode("K").mat()
        dist_coeffs = cv_file.getNode("dist_coeffs").mat()
        # read and undistort the images
        images = utils.read_captured_images()  # read all images inside the folder "data/capture"
        for img in images:
            img.undistort(K, dist_coeffs)
        image = [images[0]]  # take the first image for the extrinsic calibration

        # perform the extrinsic calibration and get the pose of the board to the camera
        # detect all chessboards and get the object and image points
        obj_pts, img_pts = calib.detect_chessboard(image, num_x, num_y, square_size)
        R, t = calib.calibrate_extrinsic(obj_pts, img_pts, K, dist_coeffs)
        print("R: ", R)
        print("t: ", t)

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 calib.py

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

# return object and image points
def detect_chessboard(images, num_x, num_y, square_size, show_images=True):
    # termination criteria
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # Arrays to store object points and image points from all the images.
    obj_points = []  # 3d point in real world space
    img_points = []  # 2d points in image plane.

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
    obj_p = np.zeros((num_y * num_x, 3), np.float32)
    obj_p[:, :2] = np.mgrid[0:num_x, 0:num_y].T.reshape(-1, 2)
    obj_p *= square_size  # add the square size of the chessboard

    for img in images:
        image_data = img.data

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

        # Find the chess board corners and append them to the point lists

        # ... end

    # return object and image points
    return obj_points, img_points

def calibrate_intrinsic(obj_points, img_points, img):
    # start ...
    print("Need to be implemented")
    # start the calibration

    # ... end

    # return camera matrix K and the distortion parameters
    return K, dist_coeffs

def calibrate_extrinsic(obj_points, img_points, K, dist_coeffs):
    # start ...
    print("Need to be implemented")
    # start the calibration

    # ... end

    # return rotation matrix R and the translation vector t
    return R[0], tvecs
```