1 Angabe Kamerakalibrierung

Führen Sie eine Kamerakalibrierung durch.

Die Bilder, welche für die Kalibrierung benötigt werden, finden Sie im Ordner ëxercise 1/images". Das Bild, welches Sie kalibrieren möchten ist ein Live Bild von der Kamera des Puzzlebots. Um dieses abzurufen verwenden Sie bitte das Hilfsprogramm `getpicture` (Hilfsprogramm-getpicture). Dieses Programm speichert das Kamerabild als `image.png` ab.

```
#!/usr/bin/env python3

from opcua import ua, uamethod, Client

user = "puzzlebot"
password = "password"
CREDENTIALS = "{}:{}".format(user, password)
hostname = "engine.ie.technikum-wien.at"
port = 4840
CLIENT = Client("opc.tcp://{}@{}:{}/".format(CREDENTIALS, hostname, port))
try:
    CLIENT.connect()
    NODE = CLIENT.get_node("ns=4;i=324")
    frame = NODE.get_value()
    f = open("image.png", "wb")
    f.write(frame)
    f.close()
finally:
    CLIENT.disconnect()
```

Quellcode 1: Hilfsprogramm-getpicture

Mit den nun vorhandenen Bildern können Sie die Kamera kalibrieren und ein Bild unverzerrt darstellen. Zunächst müssen einige Variablen angelegt werden.

- Variable für die Kantenlänge einer Kachel des Schachbretts (15mm)
- Ein Array für in den Bildern gefundene Bildpunkte
- Ein Array für in den Bildern gefundene Objektpunkte
- Eine Float32 Matrix für alle auf einem Bild enthaltenen Objektpunkte (9x8x0)

Hinweis: Erstellen Sie die Matrix mit der Funktion `numpy.zeros((9*8,3),numpy.float32)`

Befüllen Sie die Matrix mit der Funktion `numpy.mgrid()` und schneiden Sie diese mit

`*.T.reshape()` zu. Anschließend multiplizieren Sie die Matrix mit der Kachellänge des Schachbretts, um die richtige Form des Schachbretts als virtuelles Koordinatensystem zu erhalten.

Hinweis: Falls Sie mit Python Slicing nicht zu vertraut sind, verwenden Sie die folgende Zeile: `Matrix[:,:2] = numpy.mgrid[0:9,0:8].T.reshape(-1,2)* Kachellänge`

Nun können Sie mit der Funktion `glob.glob("Ordner/*jpg")` einen Bereich festlegen, in welchem die Bilder für die Kalibrierung enthalten sind. Der nächste Schritt besteht darin, jedes der Bilder mit einer Schleifenkonstruktion, Schritt für Schritt einzulesen und diese nach einem Schachbrettmuster zu überprüfen. Mit der Funktion `cv2.imread("image.jpg")` kann ein Bild eingelesen werden.

Da die Bilder ausschließlich in Graustufen verarbeitet werden können, muss das Bild in den Graustufen-Farbraum konvertiert werden. Konvertieren Sie dies mit der Funktion `cv2.cvtColor(bild,Parameter)`, wobei der Parameter für Graustufen `cv2.COLOR_BGR2GRAY` ist.

Nun beginnt die eigentliche Suche nach dem Schachbrett und dessen Kanten. Mit der Funktion `cv2.findChessboardCorners(bild,(DimensionX,DimensionY),Parameter + ...)` finden Sie, falls im jeweiligen Bild eines existiert, ein Schachbrett mit einer bestimmten Dimension. Sie verwenden für diese Übung ein Schachbrett der Dimension (9x8). Für die Parameter die diese Funktion annimmt, empfehlen sich `cv2.CALIB_CB_ADAPTIVE_THRESH` und `cv2.CALIB_CB_NORMALIZE_IMAGE` zu verwenden. Falls der Vorgang der Schachbrettmustererkennung sehr lange dauert kann auch der Parameter `cv2.CALIB_CB_FAST_CHECK` verwendet werden. Bitte beachten Sie, dass die Funktion zwei Werte zurückgibt. Der erste Wert ist vom Typ Boolean und gibt an, ob die Funktion erfolgreich war oder nicht. Der zweite Wert gibt die gefundenen Ecken zurück, wenn der erste Wert wahr ist.

Für den nächsten Schritt werden die im Schachbrett gefundenen Ecken genauer erkannt. Dazu muss allerdings überprüft werden, ob im ersten Schritt ein Schachbrett gefunden wurde. Bei Bildern, bei denen dies der Fall ist, können mit der Funktion `cv2.cornerSubPix(bild, gefundene Ecken, Größe des Suchfensters; z.B.: (5,5), Größe der Totzone zB.: (-1,-1)= keine Totzone, Parameter für Abbruchkriterium)` die Ecken verfeinert werden. Die Parameter für das Abbruchkriterium dieser Funktion sind `cv2.TERM_CRITERIA_EPS` und `cv2.TERM_CRITERIA_COUNT`. Diese Parameter beschreiben einmal die gewünschte Genauigkeit der Berechnung der Ecke, sowie die Anzahl der Berechnungen, nach welchen der Vorgang abgebrochen werden soll.

Nun haben Sie die Bildpunkte zu den Objektpunkten gefunden. Sie müssen diese nunmehr zu dem am Anfang erstellten Array für Bildpunkte hinzufügen. Dazu verwenden Sie die Funktion `Array.append(gefundene Ecken)`. Sie müssen außerdem für jedes valide Bild auch eine Matrix von Objektpunkten in das Array für die Objektpunkte hinzufügen. Für jedes Bild gibt es somit einen Eintrag im Array für Bildpunkte und Objektpunkte.

Als nächsten Schritt könnten Sie optional die Erkennung der Kanten auch noch visualisieren lassen. Dazu bietet sich die Funktion `cv2.drawChessboardCorners(bild, (DimensionX,DimensionY), gefundene Ecken, True/False)` an. Diese Funktion gibt ein Bild mit markierten Ecken an, wie in Abbildung 1 veranschaulicht ist.

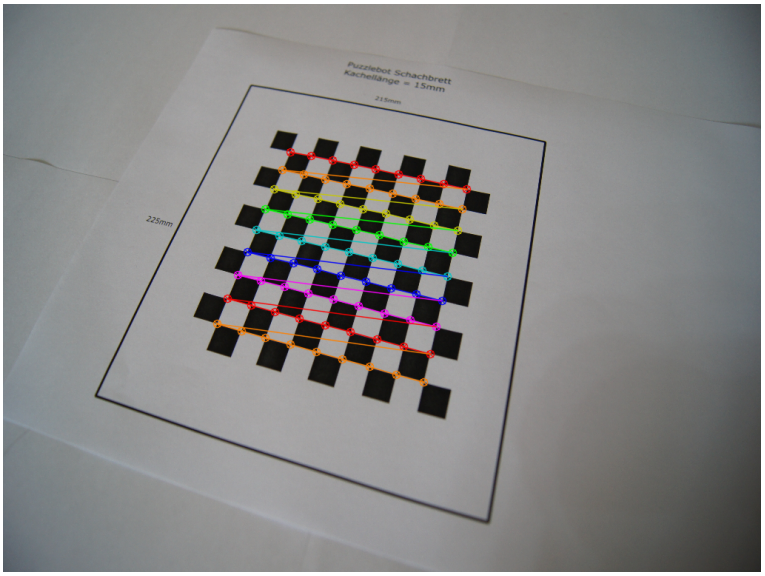

Abbildung 1: Gefundene Ecken am Schachbrett

Mit der Funktion `cv2.imshow("Fensterbezeichung",bild)` kann ein Bild in einem eigenen Fenster dargestellt werden. Das Fenster bleibt so lange geöffnet, wie Sie in der Funktion `cv2.waitKey(Zeit in Millisekunden)` definiert. Schließlich beenden Sie die geöffneten Fenster mit der Funktion `cv2.destroyAllWindows()`.

Somit haben Sie alle benötigten Werte, um nun eine Kamerakalibrierung durchzuführen. Diese wird mit der Funktion `cv2.calibrateCamera(Objektpunktearray, Bildpunktearray, Auflösung der Bilder,Parameter,Abbruchkriterium)` ausgeführt. Die Auflösung der Bilder kann als Wert angegeben werden, wenn dieser bekannt ist oder mit der Funktion `bild.shape[::-1]` abgerufen werden. Es können verschiedene Parameter angegeben werden, welche Sie aber für diese Übung nicht benötigen. Weitere Informationen sind in der OpenCV-Dokumentation zu finden. Auch ein Abbruchkriterium benötigen Sie nicht, da Ihnen nur eine begrenzte Anzahl an Bildern zur Verfügung stehen. Für beide Werte tragen Sie bitte in der Funktion `None` ein. Die Funktion der Kamerakalibrierung gibt insgesamt fünf Werte zurück. Wichtig für Sie ist der Boolean, damit Sie wissen ob die Funktion erfolgreich war oder nicht. Ebenso essenziell sind die Kameramatrix und die Verzerrungskoeffizienten. Die letzten zwei Werte sind Vektoren, welche rotatorische und translatorische Bewegungen der Bilder approximieren, und hier vernachlässigt werden können.

Mit der nun vorhandenen Kameramatrix und den Verzerrungskoeffizienten können neue Bilder entzerrt und richtig dargestellt werden. Die Kalibrierbilder liegen in einer Auflösung von

`1920x1080` Pixel vor. Hingegen lassen sich die Live-Bilder vom Puzzlebot nur in einer Auflösung von `640x480` Pixel abrufen. Die Kameramatrix ist allerdings für die höhere Auflösung gedacht und muss zuerst skaliert werden, wie in Formel 1 zu sehen ist.

$$\text{Neue Kameramatrix} = \text{Skalierung} \cdot \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Wichtig hierbei ist, dass die Kameramatrix in Python in einem Array dargestellt wird, welches einfach mit einem Faktor multipliziert werden kann. Nach der Multiplikation müssen Sie allerdings das Element [2][2] der Matrix wieder mit einer Eins ersetzen, da die letzte Zeile nicht von der Multiplikation betroffen sein darf.

Skalieren Sie die Matrix von `1920` Pixel um den Faktor 3 hinab auf `640` Pixel. Skalieren Sie, um das Seitenverhältnis zu korrigieren, den Parameter c_y um 1.333 nach oben.

Nach der Skalierung der Matrix wird nun das Kamerabild des Puzzlebots entstört. Mit der Funktion `cv2.getOptimalNewCameraMatrix(Kameramatrix,Verzerrungskoeffizienten,(Bild Breite, Bild Höhe),Parameter,Mitte des Principal Points)` wird die Kameramatrix noch auf das Bild besser angepasst. Die Funktion gibt eine neue optimale Kameramatrix zurück. Der Parameter der Funktion ist ein freier Parameter und kann zwischen 0 und 1 beliebig gewählt werden. Wenn der Parameter auf null gesetzt wird schneidet die Funktion ungewollte Pixel am Rand ab. Diese sind durch die Verzerrung entstanden.

Mit der Funktion `cv2.undistort(Bild, Kameramatrix, Verzerrungskoeffizienten, None, neue optimale Kameramatrix)` wird nun das Bild entsprechend der Kamerakalibrierung entstört.

Verwenden Sie die in Kapitel 1.1 angeführten Bibliotheken und Funktion zum Programmieren der Aufgabe 1.

1.1 Funktionen und Bibliotheken Übung 1

Bibliotheken:

- `import cv2`
- `import numpy`
- `import glob`

Funktionen:

- `glob.glob()` Pfad für Bilder erstellen

- `numpy.zeros()` Erstellen eines Array befüllt mit der Zahl 0
- `cv2.imread()` Ein Bild einlesen
- `cv2.imshow()` Ein Bild in einem Fenster darstellen
- `cv2.waitKey()` Fenster des Bildes x Millisekunden anzeigen
- `cv2.destroyAllWindows()` Fenster des Bildes schließen
- `cv2.resize()` Bild in Höhe und Breite verändern
- `cv2.cvtColor()` Konvertieren des Bildes in anderen Farbraum
- `cv2.findChessboardCorners()` Ecken des Schachbretts finden
- `cv2.cornerSubPix()` Gefundene Ecken verfeinern
- `cv2.drawChessboardCorners()` Gefundene Ecken am Schachbrett visualisieren
- `cv2.calibrateCamera()` Kamera kalibrieren
- `cv2.getOptimalNewCameraMatrix()` Kameramatrix anpassen
- `cv2.undistort()` Bild entstören