1 Task Creation of a Coordinate System and Object Localization

Create a digital coordinate system and link found objects to the calibrated image.

The third task is divided into two parts. In the first part of the exercise you will recognize corners. In the second part you will calculate a ratio to determine distances. For the first part of the third task you will need a calibrated image on which objects have already been found. To do this, you will first complete tasks one and two. When you have done this, import an image with the function `cv2.imread("image.png")` and convert it to the grayscale color space with `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` Additionally convert the image with the function `numpy.float32(image)` to the floating point 32-bit format, because the following function can only work with this format.

For the next steps you need some auxiliary variables. In this exercise you will find the workspace of the puzzle bot in an image, and describe it by known dimensions. To do this, you will recognize all corners of the workspace as shown in figure 1. These corners must be stored temporarily in an array. Create an array in which all valid corners can be stored.

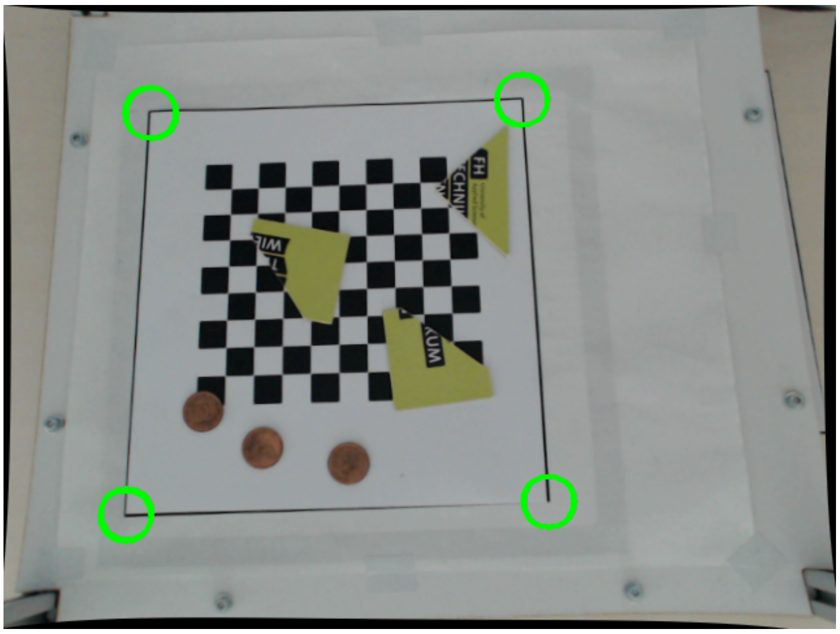

Figure 1: Corners of the workspace

With the function `cv2.cornerHarris(image, parameter blocksize, K-size, parameter k)` corners can be found. The function returns an image with values for the probability of a corner. The parameter block size describes the neighborhood (neighboring pixels), which should be examined. The K-size is passed to the function `sobel()` and describes the Sobel operator that is responsible for calculating the image derivation. The parameter k is a free parameter that influences the accuracy of the corner detection.

Note: Set the block size to 2 and the K-size to 3. Try different settings for the parameter k.

In the next step you edit the image of the `cv2.cornerHarris()` Function pixel by pixel. Create two loops. Define an area in which corners should be considered valid. Limit the area as closely as possible. However, even with a repeated camera shot of the puzzlebot (with a possibly minimal change) the area should still be considered valid. Define a parameter for the probability of a corner detection. All values above this value will be recognized as corner found.

Finally save all values into a list with the function `list.append(values)` Think about how you want to deal with duplicate values or values that are very close together. You need exactly one value per corner, which describes them. Corner detection can work very well by setting the parameters correctly, but often a corner is detected four or five times. With the function `cv2.circle()` you can graphically mark the found corners with circles.

For the second part of the exercise you need the center of gravity coordinates of a recognized coin or a recognized puzzle piece of task 2. First you choose two corners from which you know the distance to each other.
The working space of the puzzle bot is known as $215mm$ in X $\cdot$ $225mm$ in Y. So you can choose to calculate the distance on the X-axis or the Y-axis. Calculate the distance between the two corners (points) in pixels.

Hint: The origin of the coordinate system is in the upper right corner.

Save this value and next read the coordinates of the center of gravity from the desired object. Make sure that you treat the X coordinate and the Y coordinate separately, since the puzzlebot is programmed for an input of X and Y coordinate. With these values you can now set up a relationship between reference length and center of gravity coordinates. You will then get the distance in X and Y direction that the Puzzlebot must travel from the origin. You will need these values in Exercise 4 to direct the Puzzlebot to the object.

You have now completed Exercise 3. Use the libraries and functions listed in chapter 1.1 to program task 3. Please note that functions from tasks 1 and 2 can also be used.

1.1 Functions and Libraries Exercise 3

Libraries:

- `import cv2`
- `import numpy`

Functions:

- `numpy.float32()` Creating a 32bit-float data type
- `append()` Add element to a list
- `cv2.cornerHarris()` Detect a corner with Harris edge detector