hugomcp@di.ubi.pt, 2023/24

Ficha Prática 6

Camera Calibration

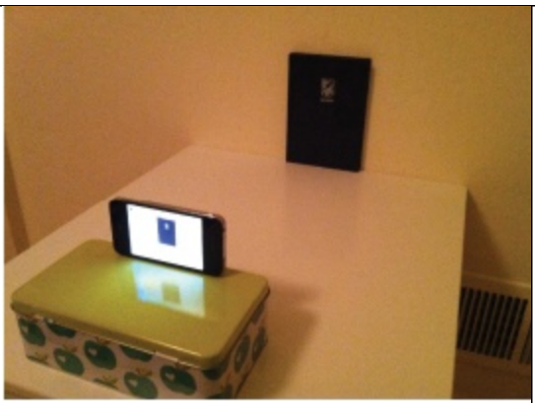
In this practical sheet, the goal is to learn calibrating a camera 1) using the simplest setup described in the tehrethical classes; and then 2) using Zhang's calibration method, implemented in OpenCV.

1. In this simplest camera calibration procedure we just need a flat object with a simple form (e.g., square or rectangle) and a ruler to obtain its physical dimensions.

   The key factor to be determined is the focal length because most of the remaining parameters can be estimated using reasonable assumptions

   Square straight pixels, optical center at the image center

   The setup is simple. We place the camera parallel to the object, with the center of mass of the object appearing near the center of the image.

	
a) Scene	b) Image

   a) Using the ruler, take a measurement of the width and height of the object (e.g., book).

      Let these values be denoted by $\mathbf{w_w}$ and $\mathbf{h_w}$

   b) Using the ruler, measure the distance between the camera and the center of mass of the object.

      Let that value be denoted by $\mathbf{d}$

c) Capture an image of the object, using the camera, similar to image b).

d) Obtain the width and height of the object in the image, measured in pixels.

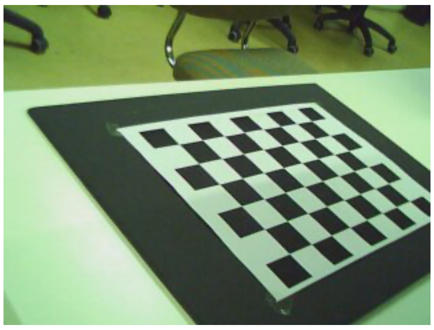
Let these values be denoted by $\mathbf{w_i}$ and $\mathbf{h_i}$

e) The camera focal length is given by:

$$f_x = \frac{w_i}{w_w} d \text{ and } f_y = \frac{h_i}{h_w} d$$

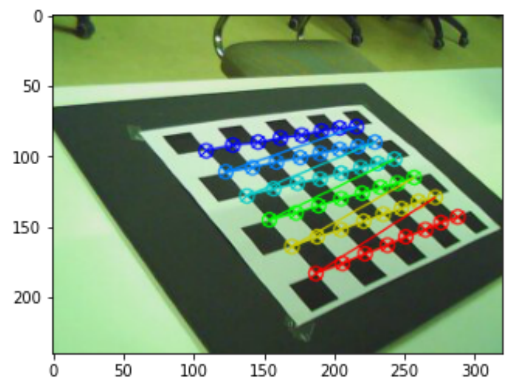
In the following exercise, we will work with the images of the "input_images_calibration" folder. This sequence contains a series of images of the calibration template. Consider that the size of each square is 30 mm, in both axes.

2. Implement the load_images(filenames) function that receives a list of image filenames and returns them as NumPy arrays.



3. Use the function "*cv2.findChessboardCorners*", and optionally "*cv2.cornerSubPix*", to automatically detect the calibration pattern and its corners in all loaded images. The calibration stencil in the images has 8 corners (rows dimension) and 6 corners (in columns dimension). It stores the results of the multiple calls in a list, so that the element ith in that list corresponds to the result of "*cv2.findChessboardCorners*" for the ith image.

4. Use "*cv2.drawChessboardCorners*" to draw the corners detected in the previous exercise. Filter out (manually) the images that were correctly detected. Ignore the rest. Display some of the resulting images.



To calibrate the camera, in addition to the coordinates of the corners in each image, the three-dimensional coordinates of the corners in the reference system of the scene should be given.

For this purpose, we consider that the center of the reference system, i.e. the point of coordinates $[0,0,0]^T$ is the first corner of the calibration template detected in the scene's reference frame. We also consider that the X-axis corresponds to the shorter side of the calibration template, i.e., the coordinate point, and the Y-axis to the longer side.

This arrangement implies that the Z-axis points in the upward normal direction of the calibration plane.

It is very important that the coordinates of the corners in the system of scene reference should be given in the same order as in which they were detected in each of the images.

5. Implement the function "get_chessboard_points (chessboard_shape, dx, dy)" that generates an array of size N×3 with the coordinates (x,y,z) of the corners of the calibration template in the reference system of the scene (N is the number of corners in the template). "*chessboard_shape*" is the number of points per row and by columns of the calibration template, i.e., (8, 6). dx (resp. dy) is the width (resp. high) of a square of the calibration template. For the template used in this lab, both values they are 30mm.

```
def get_chessboard_points(chessboard_shape, dx, dy):
   return [[(i%chessboard_shape[0])*dx,
              (i//chessboard_shape[0])*dy, 0]
                    for i in range(np.prod(chessboard_shape))]
```

6. Calibrate the camera using the results list of cv2.findChessboardCorners and the set of model points given by get_chessboard_points, from the previous exercise. Save the result of the calibration, intrinsic matrix and extrinsic matrices.

```
valid_corners = [cor[1] for cor in corners if cor[0]]

num_valid_images = len(valid_corners)
```

```
# Matrix with the coordinates of the corners
real_points = get_chessboard_points((8, 6), 30, 30)

# We are going to convert our coordinates list in the reference system to numpy array
object_points = np.asarray([real_points for i in range(num_valid_images)], dtype=np.float32)

# Convert the corners list to array
image_points = np.asarray(valid_corners, dtype=np.float32)

# Calibrate
rms, intrinsics, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(object_points, image_points,
imgs[1].shape[0:2], None, None)

# Calculate extrinsecs matrix using Rodrigues method on each rotation vector, and add its
translation vector
extrinsics = list(map(lambda rvec, tvec: np.hstack((cv2.Rodrigues(rvec)[0], tvec)), rvecs, tvecs))

# Save the calibration parameters
np.savez('calib_left', intrinsic=intrinsics, extrinsic=extrinsics)

# Print some outputs
print("Corners standard intrinsics:\n",intrinsics)
print("Corners standard dist_coefs:\n", dist_coeffs)
print("rms:\n", rms)
```