```python
##################################################
### CALL THE CAMERA AND CORRECT THE DISTORSION ###
##################################################

import cv2 # pip install opencv-python
import numpy as np

#############################
### RESOURCES AND CREDITS ###
#############################

# https://learnopencv.com/geometry-of-image-formation/
# https://learnopencv.com/understanding-lens-distortion/
# https://github.com/spmallick/learnopencv
# https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html

#################
### FUNCTIONS ###
#################

### Below are functions to recover a single frame from the camera. This is exactly like taking a picture with it.

def openCapture(camera_number_on_pc) :
    """
    This function opens and returns a camera *capture*. It represents the camera being active, and allows to recover frames.
    Once a program is done recovering frames from the camera, the capture *must* be closed with the following lines
    (or with the closeCapture function below) :

    capture.release()
    cv2.destroyAllWindows()
    """
    print("Opening capture ...")
    capture = cv2.VideoCapture(camera_number_on_pc)
    print("capture is opened:",capture.isOpened())

    while not capture.isOpened() :
        capture = cv2.VideoCapture(camera_number_on_pc)
        print("capture is opened:",capture.isOpened())
    print("-----")
    return capture


def closeCapture(capture) :
    capture.release()
    cv2.destroyAllWindows()
```

```
def get_frame_from_capture(capture) :
    """
    This function basically takes a picture with an open camera capture, and returns said picture.
    """

    print("Recovering frame ...")

    ret, frame = capture.read()
    print("Got a frame:", ret)

    while not ret :
        ret, frame = capture.read()
        print("Got a frame:", ret)

    # Below is a cv2-specific time.sleep(), which takes a duration in milliseconds as argument.
    # Somehow, without it, the recovered frame isn't updated and remains completely grey.
    cv2.waitKey(200)

    print("-----")
    cv2.destroyAllWindows()
    return frame


def manually_get_frame_from_capture(capture, window_name="source", flip = True) :
    """
    This function takes a picture with an open camera capture, and returns said picture, when the user
presses ESC.
    """

    while(True):
        ret, frame = capture.read()
        if ret:
            if flip :
                cv2.imshow(window_name, cv2.flip(frame,1))
            else :
                cv2.imshow(window_name, frame)
        a = cv2.waitKey(200)
        if a == 27: #ESC key
            break

    print("-----")
    cv2.destroyAllWindows()
    return frame

### Below are functions to properly calibrate a camera and remove the distorsion caused by its lens.
### More specifically, these functions will ultimately return an optimized camera matrix, which needs to be
determined only once.
### The camera matrix can be used afterwards to remove the distorsion.
```

```python
def cameraMatrixRecover(checkerboard_images, checkerboard_internal_corners) :
    # Defining the dimensions of checkerboard
    # The number of internal corners MUST BE EXACT
    checkerboard = checkerboard_internal_corners
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # Creating vector to store vectors of 3D points for each checkerboard image
    objpoints = []
    # Creating vector to store vectors of 2D points for each checkerboard image
    imgpoints = []

    # Defining the world coordinates for 3D points
    objp = np.zeros((1, checkerboard[0]*checkerboard[1], 3), np.float32)
    objp[0,:,:2] = np.mgrid[0:checkerboard[0], 0:checkerboard[1]].T.reshape(-1, 2)
    prev_img_shape = None

    # Extracting path of individual image stored in a given directory
    for img in checkerboard_images:
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        # Find the chess board corners
        # If desired number of corners are found in the image then ret = true
        ret, corners = cv2.findChessboardCorners(gray, checkerboard, cv2.CALIB_CB_ADAPTIVE_THRESH +
cv2.CALIB_CB_FAST_CHECK + cv2.CALIB_CB_NORMALIZE_IMAGE)
        print("Found chessboard :", ret)
        """
        If desired number of corner are detected,
        we refine the pixel coordinates and display
        them on the images of checker board
        """
        if ret == True:
            objpoints.append(objp)
            # refining pixel coordinates for given 2d points.
            corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)

            imgpoints.append(corners2)

            # Draw and display the corners
            img = cv2.drawChessboardCorners(img, checkerboard, corners2,ret)

            cv2.imshow('img',img)
        cv2.waitKey(0)

    cv2.destroyAllWindows()

    h,w = img.shape[:2]

    """
    Performing camera calibration by
    passing the value of known 3D points (objpoints)
    and corresponding pixel coordinates of the
    detected corners (imgpoints)
    """
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)

    # Refining the camera matrix using parameters obtained by calibration
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

    return mtx, newcameramtx, dist


def undistortImage(img, mtx, newcameramtx, dist) :
    h,w = img.shape[:2]

    # Method 1 to undistort the image
    undistorted = cv2.undistort(img, mtx, dist, None, newcameramtx)

    # Method 2 to undistort the image
    mapx,mapy=cv2.initUndistortRectifyMap(mtx,dist,None,newcameramtx,(w,h),5)
    undistorted = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)

    return undistorted



def manually_get_undistorted_frame_from_capture(capture, mtx, newcameramtx, dist, window_name="source", flip
= True) :
    while(True):
        ret, frame = capture.read()
        undistorted = undistortImage(frame, mtx, newcameramtx, dist)
        if ret:
            if flip :
                cv2.imshow(window_name, cv2.flip(undistorted,1))
            else :
                cv2.imshow(window_name, undistorted)
        a = cv2.waitKey(200)
        if a == 27: #Touche ECHAP
            break

    print("-----")
    cv2.destroyAllWindows()
    return undistorted



def liveCameraCalibration(camera_number_on_pc, checkerboard_internal_corners) :

    # Recovering checkerboard images to calibrate
    capture = openCapture(camera_number_on_pc)
    checkerboard_images = []
    for i in range(15) :
        checkerboard_images.append(manually_get_frame_from_capture(capture, "checkerboard picture
n°"+str(i+1), False))


    mtx, newcameramtx, dist = cameraMatrixRecover(checkerboard_images, checkerboard_internal_corners)

    print("Camera matrix :")
    print(mtx)
    print()
    print("Optimal new camera matrix :")
    print(newcameramtx)
    print()
    print("dist :")
    print(dist)

    for img in checkerboard_images :
        undistorted = undistortImage(img, mtx, newcameramtx, dist)
        # Displaying the undistorted image
        cv2.imshow("undistorted image",undistorted)
        cv2.waitKey(0)

    cv2.destroyAllWindows()
    manually_get_undistorted_frame_from_capture(capture, mtx, newcameramtx, dist, "source", False)
    closeCapture(capture)

    return mtx, newcameramtx, dist
```