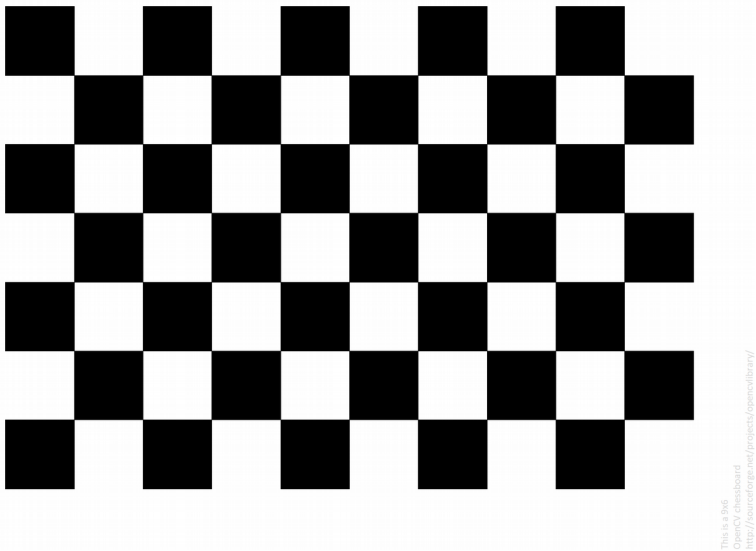
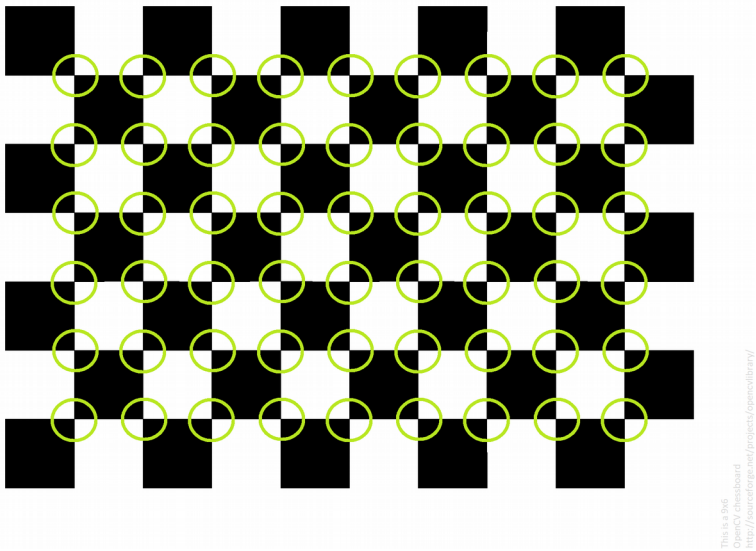
PRACTICA 2: Camera Calibration

Introducció

Una càmera es calibra usant un patró amb contrastos de colors fàcils de detectar. S'ha estandarditzat el tauler d'escacs com a patró per calibrar per la senzillesa de buscar el canvi de contrast en les cantonades dels quadrats i al fet que la major diferència de contrast està entre els colors blanc i negre. El tauler d'escacs no és l'únic tipus de patró que s'utilitza, però si el més senzill, per a aquesta pràctica utilitzarem un tauler de 9x6 com el següent (el tauler deu portar-se imprès a la pràctica en grandària A4, és l'arxiu adjunt “pattern.png”):



És un tauler 9x6 per què, encara que veiem 10x7 columnes i files de quadrats, es compten els vèrtexs on s'ajunten 2 quadrats negres amb 2 de blancs, tal com s’assenyala en la següent imatge:



Calibratge de la càmera

Cal començar per importar les biblioteques d'OpenCV i de Numpy que ja hem utilitzat a la pràctica anterior:

```
import numpy as np
import cv2
```

A continuación cal definir els criteris de terminació de sub-píxel per identificar les cantonades i iniciar els coordinadors de punts d'objectes (objp). També definirem ja els arrays per tractar els punts captats:

```
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.

objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
```

Després de localitzar un mínim de 10 frames on la càmera reconegui el patró de cantonades del tauler d'escacs, la matriu de la càmera i els coeficients de distorsió poden ser calculats mitjançant la funció calibrateCamera. Aquí tenim l'exemple de com fer-ho amb la càmera a temps real:

```
cap = cv2.VideoCapture(0)
found = 0
while(found < 10):  # Here, 10 can be changed to whatever number you like to choose
    ret, img = cap.read() # Capture frame-by-frame
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (9,6), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)  # Certainly, every loop objp is the same, in 3D.
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)
        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (9,6), corners2, ret)
        found += 1
    cv2.imshow('img', img)
    cv2.waitKey(10)
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Si instal·lem la biblioteca yaml, ens permetrà desar aquestes dades de calibratge en un arxiu afegint poques línies de codi al final del programa. En primer lloc caldria declarar la biblioteca a l'inici del programa:

```
import yaml
```

I a continuació afegir les següents línies al final del programa per convertir totes les dades de la matriu de calibratge a format llistat per guardar-los en un arxiu denominat “calibration.yaml”:

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)



# It's very important to transform the matrix to list.
data = {'camera_matrix': np.asarray(mtx).tolist(), 'dist_coeff': np.asarray(dist).tolist()}


with open("calibration.yaml", "w") as f:
   yaml.dump(data, f)
```

Si posteriorment volguéssim carregar aquests valors de calibratge desats en l'arxiu “calibration.yaml” només hauríem d'escriure les següents línies per carregar la informació:

```
with open('calibration.yaml') as f:
    loadeddict = yaml.load(f)

mtxloaded = loadeddict.get('camera_matrix')
distloaded = loadeddict.get('dist_coeff')
```

Pose estimation

Havent calibrat la càmera podem utilitzar aquest patró com a base de referència sobre la qual treballar, atès que és un patró conegut. Això ens permet representar en AR (Realitat Augmentada) els eixos X, Y i Z del nostre sistema o bé diferents figures geomètriques simples. Una vegada representades sobre el patró, si el desplacem o li canviem l'orientació davant de la càmera, veurem com l'objecte representat es mou al compàs del moviment del patró.

Llegiu i realitzeu el següent tutorial:

http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_pose/py_pose.html

FURTHER WORK: Si trobeu temps, intenteu modificar la figura que es posiciona sobre el tauler d'escacs amb una figura 3D que us agradi.

Calibratge de dues càmeres per a estereovisió

El següent codi realitza el calibratge d'un sistema d'estereovisió mitjançant dues càmeres que enfoquen al mateix patró. Com que no disposem de dues càmeres, optem per fer un codi que parteix de dues llistes d'imatges representant cada llista a una de les càmeres. En el report, comenteu aquest codi degudament, acompanyant-lo de la descripció detallada de la seva funcionalitat.

```
import numpy as np
import cv2
import glob
import argparse


class StereoCalibration(object):
    def __init__(self, filepath):
        # termination criteria
        self.criteria = (cv2.TERM_CRITERIA_EPS +
                         cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        self.criteria_cal = (cv2.TERM_CRITERIA_EPS +
                             cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-5)

        # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
        self.objp = np.zeros((9*6, 3), np.float32)
        self.objp[:, :2] = np.mgrid[0:9, 0:6].T.reshape(-1, 2)

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

        self.cal_path = filepath
        self.read_images(self.cal_path)

    def read_images(self, cal_path):
        images_right = glob.glob(cal_path + 'RIGHT/*.JPG')
        images_left = glob.glob(cal_path + 'LEFT/*.JPG')
        images_left.sort()
        images_right.sort()

        for i, fname in enumerate(images_right):
            img_l = cv2.imread(images_left[i])
            img_r = cv2.imread(images_right[i])

            gray_l = cv2.cvtColor(img_l, cv2.COLOR_BGR2GRAY)
            gray_r = cv2.cvtColor(img_r, cv2.COLOR_BGR2GRAY)

            # Find the chess board corners
            ret_l, corners_l = cv2.findChessboardCorners(gray_l, (9, 6), None)
            ret_r, corners_r = cv2.findChessboardCorners(gray_r, (9, 6), None)
```

```
            # If found, add object points, image points (after refining them)
            self.objpoints.append(self.objp)

            if ret_l is True:
                rt = cv2.cornerSubPix(gray_l, corners_l, (11, 11),
                                      (-1, -1), self.criteria)
                self.imgpoints_l.append(corners_l)

                # Draw and display the corners
                ret_l = cv2.drawChessboardCorners(img_l, (9, 6),
                                                  corners_l, ret_l)
                cv2.imshow(images_left[i], img_l)
                cv2.waitKey(500)

            if ret_r is True:
                rt = cv2.cornerSubPix(gray_r, corners_r, (11, 11),
                                      (-1, -1), self.criteria)
                self.imgpoints_r.append(corners_r)

                # Draw and display the corners
                ret_r = cv2.drawChessboardCorners(img_r, (9, 6),
                                                  corners_r, ret_r)
                cv2.imshow(images_right[i], img_r)
                cv2.waitKey(500)
            img_shape = gray_l.shape[::-1]

        rt, self.M1, self.d1, self.r1, self.t1 = cv2.calibrateCamera(
            self.objpoints, self.imgpoints_l, img_shape, None, None)
        rt, self.M2, self.d2, self.r2, self.t2 = cv2.calibrateCamera(
            self.objpoints, self.imgpoints_r, img_shape, None, None)

        self.camera_model = self.stereo_calibrate(img_shape)

    def stereo_calibrate(self, dims):
        flags = 0
        flags |= cv2.CALIB_FIX_INTRINSIC
        # flags |= cv2.CALIB_FIX_PRINCIPAL_POINT
        flags |= cv2.CALIB_USE_INTRINSIC_GUESS
        flags |= cv2.CALIB_FIX_FOCAL_LENGTH
        # flags |= cv2.CALIB_FIX_ASPECT_RATIO
        flags |= cv2.CALIB_ZERO_TANGENT_DIST
        # flags |= cv2.CALIB_RATIONAL_MODEL
        # flags |= cv2.CALIB_SAME_FOCAL_LENGTH
        # flags |= cv2.CALIB_FIX_K3
        # flags |= cv2.CALIB_FIX_K4
        # flags |= cv2.CALIB_FIX_K5

        stereocalib_criteria = (cv2.TERM_CRITERIA_MAX_ITER +
```

```
                                cv2.TERM_CRITERIA_EPS, 100, 1e-5)
        ret, M1, d1, M2, d2, R, T, E, F = cv2.stereoCalibrate(
            self.objpoints, self.imgpoints_l,
            self.imgpoints_r, self.M1, self.d1, self.M2,
            self.d2, dims,
            criteria=stereocalib_criteria, flags=flags)

        print('Intrinsic_mtx_1', M1)
        print('dist_1', d1)
        print('Intrinsic_mtx_2', M2)
        print('dist_2', d2)
        print('R', R)
        print('T', T)
        print('E', E)
        print('F', F)

        # for i in range(len(self.r1)):
        #     print("--- pose[", i+1, "] ---")
        #     self.ext1, _ = cv2.Rodrigues(self.r1[i])
        #     self.ext2, _ = cv2.Rodrigues(self.r2[i])
        #     print('Ext1', self.ext1)
        #     print('Ext2', self.ext2)

        print('')

        camera_model = dict([('M1', M1), ('M2', M2), ('dist1', d1),
                             ('dist2', d2), ('rvecs1', self.r1),
                             ('rvecs2', self.r2), ('R', R), ('T', T),
                             ('E', E), ('F', F)])

        cv2.destroyAllWindows()
        return camera_model

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('filepath', help='String Filepath')
    args = parser.parse_args()
    cal_data = StereoCalibration(args.filepath)
```