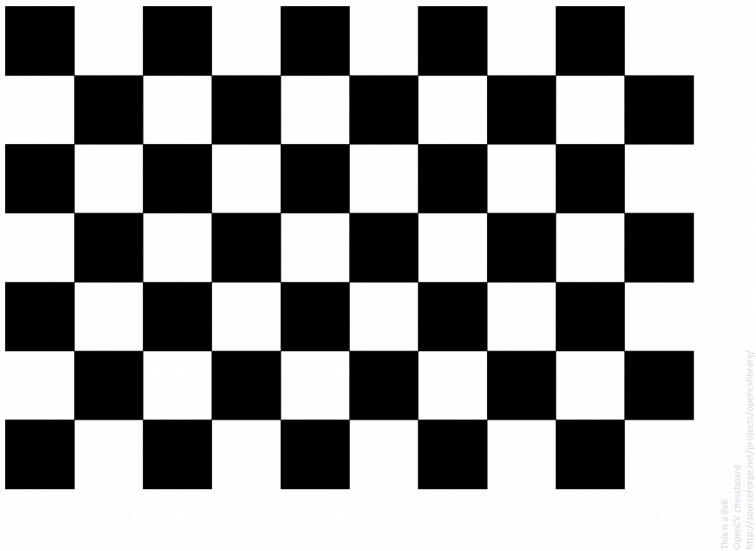
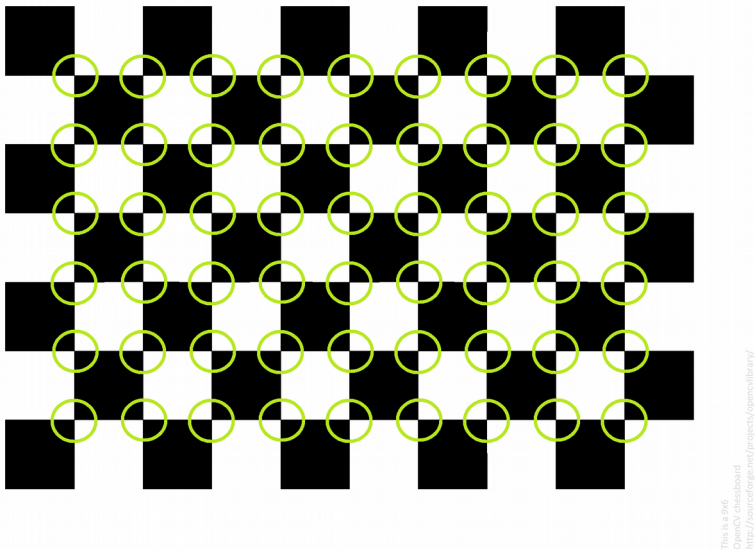
PRACTICA 2: Camera Calibration

Introducción

Una cámara se calibra usando un patrón con contrastes de colores fáciles de detectar. Se ha estandarizado el tablero de ajedrez cómo patrón para calibrar debido a la sencillez de buscar el cambio de contraste en las esquinas de los cuadrados y a que la mayor diferencia de contraste está entre los colores blanco y negro. El tablero de ajedrez no es el único tipo de patrón que se utiliza, pero si el más sencillo, para esta práctica utilizaremos un tablero de 9x6 cómo el siguiente (el tablero debe traerse imprimido a la práctica en tamaño A4, es el archivo adjunto pattern.png):



Se trata de un tablero 9x6 debido a que, aunque veamos 10x7 columnas y filas de cuadrados, se cuentan los vértices dónde se juntan 2 cuadrados negros con 2 blancos, cómo está señalado en la siguiente imagen:



Calibración de la cámara

Debemos empezar por importar las bibliotecas de OpenCV y de Numpy que ya utilizamos en la práctica anterior:

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

A continuación es necesario definir los criterios de terminación de sub-pixel para identificar las esquinas e iniciar los coordinadores de puntos de objetos (objp). También definiremos ya los arrays para tratar los puntos captados:

```
# 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.
```

Tras localizar un mínimo de 10 frames dónde la cámara reconozca el patrón de esquinas del tablero de ajedrez, la matriz de la camara y los coeficientes de distorsion pueden ser calculador mediante la función calibrateCamera. Aquí tenemos el ejemplo de cómo hacerlo con la cámara a tiempo 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 instalásemos la librería yaml, nos permitiría guardar estos datos de calibrado en un archivo añadiendo unas pocas líneas de código al final del programa. En primer lugar habría que declarar la librería al inicio del programa:

```
import yaml
```

Y a continuación añadir las siguientes líneas al final del programa para convertir todos los datos de la matriz de calibración a formato listado para guardarlos en un archivo denominado “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 posteriormente quisiéramos cargar estos valores de calibrado guardados en el archivo “calibration.yaml” solo deberíamos escribir las siguientes líneas para cargar la información:

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

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

Pose estimation

Posteriormente a haber calibrado la cámara podemos utilizar el mismo patrón cómo base de referencia sobre la que trabajar, ya que se trata de un patrón conocido. Esto nos permite representar en AR (Realidad Aumentada) los ejes X, Y y Z de nuestro sistemas o diferentes figuras geométricas simples. Una vez representadas sobre el patrón, si desplazamos este o le cambiamos la inclinación frente a la cámara veremos cómo el objeto representado se moverá de la misma manera.

Leed e implementad el siguiente tutorial:

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

FURTHER WORK: Si encontráis tiempo, intentad modificar la figura que se posiciona sobre el tablero de ajedrez con una figura 3D que os guste.

Calibración de dos cámaras para estereovisión

El siguiente código realiza la calibración de un sistema de estereovisión mediante dos cámaras que enfocan al mismo patrón. Al no disponer de dos cámaras para su realización se trata de un código que parte de dos listas de imágenes representando cada lista a una de las cámaras. En el informe, comentad este código debidamente, acompañándolo de la descripción detallada de su funcionalidad.

```
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)
```