□

Homework4 AR

胡成成 2101210578

Question

1. 请把下图中学校的名牌（或名字）换成其它任意学校名，做到与原有场景一致。（或自选图片和替换对象）

2. 用python，opencv 实现AR.开源项目 BAR4Py. https://github.com/GeekLiB/AR-BXT-AR4Python

3.基于PyOpenGL和Pygame的增强现实，https://blog.csdn.net/weixin_43842653/article/details/89071046

Answer

选择第二题：

- 由于该开源项目由Python2编写，与现在主流的Python3存在很多差异，需要对代码的一些细节进行修改才能运行
- getMatrix

```
# -*- coding:utf-8 -*-
import cv2
import numpy as np


class GetPMatrix:
    """
    findMark, getMatches: 寻找标记物
    getP: 反馈相机外参、内参以及畸变系数
    getGLP, getGLM: 分别反馈OpenGL形式的投影矩阵和视图矩阵

    """

    def __init__(self, markImage, MIN_MATCH_COUNT=10, PCount=20, DequeLen=5):
        """
        markImage: 标记图片的array形式
        MIN_MATCH_COUNT: 最小优越点数目
        PCount: getP 执行第PCount，停止内参标定
        DequeLen: 遗忘队列长度

        """
        # Init MarkImage, Debug.
        h, w = markImage.shape[:2]
        if w > h:
            t = int(float(w - h) / 2)
            self.MarkImage = markImage[:h, t:t + h]
        else:
            t = int(float(h - w) / 2)
            self.MarkImage = markImage[t:t + w, :w]
        self.MIN_MATCH_COUNT = MIN_MATCH_COUNT

        self.SceneImage = None
        self.DrawParams = None
        self.KP1 = None
        self.KP2 = None
        self.GoodMatches = None

        from collections import deque
        self.PTimes = 0
        self.PCount = PCount
        self.OBJPoints = deque(maxlen=DequeLen)
        self.IMGPoints = deque(maxlen=DequeLen)
        self.MTX = None
        self.DIST = None
        self.RVEC = None
        self.TVEC = None

    def findMark(self, sceneImage, pdLimit=16, hdLimit=10):
        """
        sceneImage: 场景图片的array形式
        pdLimit: 四边形轮廓最小边距
        hdLimit: 图片最大hash误差

        return outDst: 反馈标记物关键点

        """

        # Defined functions.
        def isGoodApprox(approx, limit):
            if approx.shape != (4, 1, 2):
                return False
            for i in range(4):
                distance = np.sqrt(np.sum((approx[i] - approx[(i + 1) % 4]) ** 2))
                if distance < limit:
                    return False
            return True

        def puzzleMark(dst, mark, limit):
            def getImageHash(img):
                img2 = cv2.resize(img, (8, 8), interpolation=cv2.INTER_CUBIC)
                img3 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
                ret, thresh = cv2.threshold(img3, img3.mean(), 255, 0)
                imgHash = np.zeros(shape=(8, 8), dtype=np.int8)
                imgHash[thresh > 0] = 1
                return imgHash

            def rotationHash(_hash):
                outHash = np.zeros(shape=(8, 8), dtype=np.int8)
                for i in range(8):
                    for j in range(8):
                        outHash[7 - j, i] = _hash[i, j]
                return outHash

            dstHash, markHash = getImageHash(dst), getImageHash(mark)

            for i in range(4):
                hashDistance = np.sum(np.abs((dstHash - markHash)))
                if hashDistance < limit:
                    return i + 1
                markHash = rotationHash(markHash)
            else:
                return 0

        imgray = cv2.cvtColor(sceneImage, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(imgray, 127, 255, 0)
        contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

        for cnt in contours:
            epsilon = 0.05 * cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, epsilon, True)
            if isGoodApprox(approx, pdLimit):
                pts1 = np.float32(approx[:, 0, :])
                pts2 = np.float32([[0, 0], [0, 64], [64, 64], [64, 0]])

                M = cv2.getPerspectiveTransform(pts1, pts2)
                dst = cv2.warpPerspective(sceneImage, M, (64, 64))

                tag = puzzleMark(dst, self.MarkImage, hdLimit)
                outDst = np.zeros(shape=(4, 1, 2), dtype=np.float32)
                if tag:
                    for i in range(4):
                        outDst[i, 0, :] = approx[(i + (tag - 1)) % 4, 0, :]

                    self.SceneImage = sceneImage
                    return outDst
                else:
                    return None

    def getMatches(self, sceneImage):
        """
        sceneImage: 场景图片的array形式

        return dst: 反馈标记物关键点

        """
        # Initiate SIFT detector
        sift = cv2.xfeatures2d.SIFT_create()

        # find the keypoints and descriptors with SIFT
        kp1, des1 = sift.detectAndCompute(self.MarkImage[:, :, 0], None)
        kp2, des2 = sift.detectAndCompute(sceneImage[:, :, 0], None)

        # create BFMatcher object
        FLANN_INDEX_KDTREE = 0
        index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
        search_params = dict(checks=50)

        flann = cv2.FlannBasedMatcher(index_params, search_params)

        # Match descriptors.
        matches = flann.knnMatch(des1, des2, k=2)

        # Sort them in the order of their distance.
        good = []
        for m, n in matches:
            if m.distance < 0.7 * n.distance:
                good.append(m)
        if len(good) < self.MIN_MATCH_COUNT:
            return None

        src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        matchesMask = mask.ravel().tolist()

        h, w = self.MarkImage.shape[:2]
        pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
        dst = cv2.perspectiveTransform(pts, M)

        draw_params = dict(matchColor=(0, 255, 0),  # draw matches in green color
                           singlePointColor=None,
                           matchesMask=matchesMask,  # draw only inliers
                           flags=2)

        self.SceneImage = sceneImage
        self.DrawParams = draw_params
        self.KP1 = kp1
        self.KP2 = kp2
        self.GoodMatches = good
        return dst

    def getP(self, dst):
        """
        dst: 标记物关键点

        return self.MTX,self.DIST,self.RVEC,self.TVEC:
        反馈 内参、畸变系数，旋转向量，位移向量

        """
        if self.SceneImage is None:
            return None

        corners = np.float32([dst[1], dst[0], dst[2], dst[3]])
        gray = cv2.cvtColor(self.SceneImage, cv2.COLOR_BGR2GRAY)
        # 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), (1,0,0), (1,1,0)
        objp = np.zeros((2 * 2, 3), np.float32)
        objp[:, :2] = np.mgrid[0:2, 0:2].T.reshape(-1, 2)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

        if self.PTimes < self.PCount or self.PCount == 0:
            # Arrays to store object points and image points from all the images.
            objpoints = self.OBJPoints  # 3d point in real world space
            imgpoints = self.IMGPoints  # 2d points in image plane.

            if len(imgpoints) == 0 or np.sum(np.abs(imgpoints[-1] - corners2)) != 0:
                objpoints.append(objp)
                imgpoints.append(corners2)

            # Find mtx, dist, rvecs, tvecs
            ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
            if not ret:
                self.PTimes += 1
                return None
            self.OBJPoints = objpoints
            self.IMGPoints = imgpoints
            self.MTX = mtx
            self.DIST = dist
            self.RVEC = rvecs[0]
            self.TVEC = tvecs[0]
        else:
            # Find the rotation and translation vectors.
            _, rvec, tvec, _ = cv2.solvePnPRansac(objp, corners2, self.MTX, self.DIST)
            self.RVEC = rvec
            self.TVEC = tvec
        self.PTimes += 1

        return self.MTX, self.DIST, self.RVEC, self.TVEC

    def getGLP(self, width, height):
        """
        width, height: 场景宽、高
        """
        P = np.zeros(shape=(4, 4), dtype=np.float32)

        fx = self.MTX[0, 0]
        fy = self.MTX[1, 1]

        cx = self.MTX[0, -1]
        cy = self.MTX[1, -1]

        near = 0.1
        far = 100.0

        P[0, 0] = 2 * fx / width
        P[1, 1] = 2 * fy / height
        P[0, 2] = 1 - (2 * cx / width)
        P[1, 2] = (2 * cy / height) - 1
        P[2, 2] = -(far + near) / (far - near)
        P[3, 2] = -1.
        P[2, 3] = -(2 * far * near) / (far - near)

        p = P.T
        return p.flatten()

    def getGLM(self):
        R, _ = cv2.Rodrigues(self.RVEC)
        Rt = np.hstack((R, self.TVEC))

        Rx = np.array([[1, 0, 0], [0, -1, 0], [0, 0, -1]])

        M = np.eye(4)
        M[:3, :] = np.dot(Rx, Rt)

        m = M.T
        return m.flatten()

    # Debug code.
    def drawMatches(self, MarkImage, SceneImage):
        outImg = cv2.drawMatches(MarkImage, self.KP1,
                                 SceneImage, self.KP2,
                                 self.GoodMatches, None, **self.DrawParams)
        return outImg

    def drawBox(self, img):
        axis = np.float32([[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0],
                           [0, 0, -1], [0, 1, -1], [1, 1, -1], [1, 0, -1]])
        imgpts, jac = cv2.projectPoints(axis, self.RVEC, self.TVEC, self.MTX, self.DIST)
        imgpts = np.int32(imgpts).reshape(-1, 2)

        # draw pillars in blue color
        for i, j in zip(range(4), range(4, 8)):
            img2 = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]), (255, 0, 0), 3)

        # draw top layer in red color
        outImg = cv2.drawContours(img2, [imgpts[4:]], -1, (0, 0, 255), 3)

        return outImg


# Debug Code.
def debugMark():
    # Debug module.
    # from matplotlib import pyplot as plt

    markImage = cv2.imread('mark.png')
    sceneImage = cv2.imread('mark_in_scene.png')
    # Init PM.
    pm = GetPMatrix(markImage)

    # Get kp1, kp2, dst, goodMatches, [draw_params].
    dst = pm.findMark(sceneImage)
    if dst is None:
        exit()

    # # Get ret, mtx, dist, rvecs, tvecs
    tmp = None
    for i in range(30):
        tmp = pm.getP(dst)
        if tmp is None:
            exit()
        # print i
    mtx, dist, rvec, tvec = tmp

    # Draw Box
    h, w = markImage.shape[:2]
    sceneImage = pm.drawBox(sceneImage)

    # Draw corners.
    for point in dst:
        cv2.circle(sceneImage, tuple(point[0]), 5, (0, 0, 255), -1)

    h2, w2 = sceneImage.shape[:2]
    glP = pm.getGLP(w2, h2)
    glM = pm.getGLM()


    markImage = cv2.cvtColor(markImage, cv2.COLOR_BGR2RGB)
    sceneImage = cv2.cvtColor(sceneImage, cv2.COLOR_BGR2RGB)
    plt.figure('Mark test.')
    plt.subplot(121), plt.imshow(markImage), plt.title('Mark')
    plt.subplot(122), plt.imshow(sceneImage), plt.title('Scene')


def debugMatches():
    # Debug module.
    # from matplotlib import pyplot as plt

    markImage = cv2.imread('clock.png')
    sceneImage = cv2.imread('clock_in_scene.png')

    # Init PM.
    pm = GetPMatrix(markImage)

    # Get kp1, kp2, dst, goodMatches, [draw_params].
    dst = pm.getMatches(sceneImage)
    if dst is None:
        exit()

    # Draw circles and lines.
    img3 = pm.drawMatches(markImage, sceneImage)

    # # Get ret, mtx, dist, rvecs, tvecs
    tmp = None
    for i in range(30):
        tmp = pm.getP(dst)
        if tmp is None:
            exit()
        # print i
    mtx, dist, rvec, tvec = tmp

    # Draw Box
    h, w = markImage.shape[:2]
    img3[:, w:] = pm.drawBox(img3[:, w:])

    h2, w2 = sceneImage.shape[:2]
    glP = pm.getGLP(w2, h2)
    glM = pm.getGLM()

    print('mtx -------------')
    print(mtx)
    print('dist ------------')
    print(dist)
    print('rvec -----------')
    print(rvec)
    print('tvec -----------')
    print(tvec)
    print('glP ------------')
    print(glP)
    print('glM ------------')
    print(glM)

    img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
    plt.figure('Matches test.'), plt.imshow(img3)


if __name__ == '__main__':
    # Debug module.
    from matplotlib import pyplot as plt

    debugMark()
    debugMatches()
    plt.show()
```

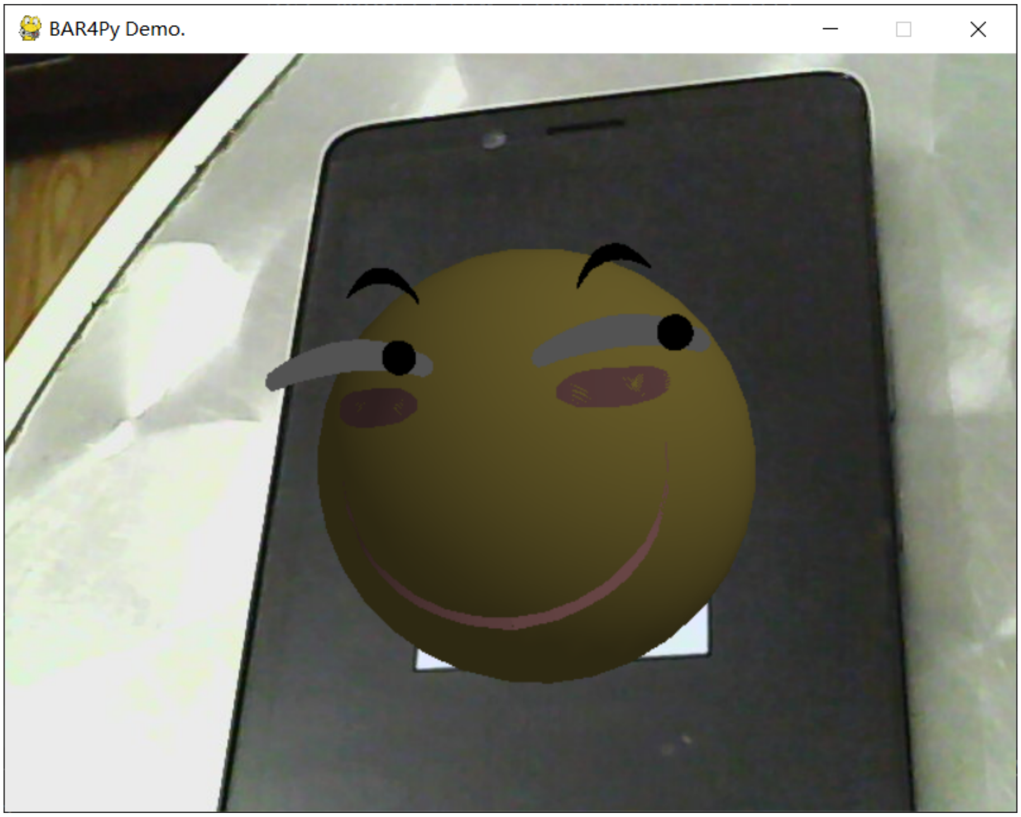
- objloader

```
# -*- coding:utf-8 -*-
import cv2
import numpy as np


class GetPMatrix:
    """
    findMark, getMatches: 寻找标记物
    getP: 反馈相机外参、内参以及畸变系数
    getGLP, getGLM: 分别反馈OpenGL形式的投影矩阵和视图矩阵

    """

    def __init__(self, markImage, MIN_MATCH_COUNT=10, PCount=20, DequeLen=5):
        """
        markImage: 标记图片的array形式
        MIN_MATCH_COUNT: 最小优越点数目
        PCount: getP 执行第PCount，停止内参标定
        DequeLen: 遗忘队列长度

        """
        # Init MarkImage, Debug.
        h, w = markImage.shape[:2]
        if w > h:
            t = int(float(w - h) / 2)
            self.MarkImage = markImage[:h, t:t + h]
        else:
            t = int(float(h - w) / 2)
            self.MarkImage = markImage[t:t + w, :w]
        self.MIN_MATCH_COUNT = MIN_MATCH_COUNT
        self.SceneImage = None
        self.DrawParams = None
        self.KP1 = None
        self.KP2 = None
        self.GoodMatches = None

        from collections import deque
        self.PTimes = 0
        self.PCount = PCount
        self.OBJPoints = deque(maxlen=DequeLen)
        self.IMGPoints = deque(maxlen=DequeLen)
        self.MTX = None
        self.DIST = None
        self.RVEC = None
        self.TVEC = None

    def findMark(self, sceneImage, pdLimit=16, hdLimit=10):
        """
        sceneImage: 场景图片的array形式
        pdLimit: 四边形轮廓最小边距
        hdLimit: 图片最大hash误差

        return outDst: 反馈标记物关键点

        """

        # Defined functions.
        def isGoodApprox(approx, limit):
            if approx.shape != (4, 1, 2):
                return False
            for i in range(4):
                distance = np.sqrt(np.sum((approx[i] - approx[(i + 1) % 4]) ** 2))
                if distance < limit:
                    return False
            return True

        def puzzleMark(dst, mark, limit):
            def getImageHash(img):
                img2 = cv2.resize(img, (8, 8), interpolation=cv2.INTER_CUBIC)
                img3 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
                ret, thresh = cv2.threshold(img3, img3.mean(), 255, 0)
                imgHash = np.zeros(shape=(8, 8), dtype=np.int8)
                imgHash[thresh > 0] = 1
                return imgHash

            def rotationHash(_hash):
                outHash = np.zeros(shape=(8, 8), dtype=np.int8)
                for i in range(8):
                    for j in range(8):
                        outHash[7 - j, i] = _hash[i, j]
                return outHash

            dstHash, markHash = getImageHash(dst), getImageHash(mark)

            for i in range(4):
                hashDistance = np.sum(np.abs((dstHash - markHash)))
                if hashDistance < limit:
                    return i + 1
                markHash = rotationHash(markHash)
            else:
                return 0

        imgray = cv2.cvtColor(sceneImage, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(imgray, 127, 255, 0)
        contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
        for cnt in contours:
            epsilon = 0.05 * cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, epsilon, True)
            if isGoodApprox(approx, pdLimit):
                pts1 = np.float32(approx[:, 0, :])
                pts2 = np.float32([[0, 0], [0, 64], [64, 64], [64, 0]])

                M = cv2.getPerspectiveTransform(pts1, pts2)
                dst = cv2.warpPerspective(sceneImage, M, (64, 64))

                tag = puzzleMark(dst, self.MarkImage, hdLimit)
                outDst = np.zeros(shape=(4, 1, 2), dtype=np.float32)
                if tag:
                    for i in range(4):
                        outDst[i, 0, :] = approx[(i + (tag - 1)) % 4, 0, :]

                    self.SceneImage = sceneImage
                    return outDst
                else:
                    return None

    def getMatches(self, sceneImage):
        """
        sceneImage: 场景图片的array形式

        return dst: 反馈标记物关键点

        """
        # Initiate SIFT detector
        sift = cv2.xfeatures2d.SIFT_create()

        # find the keypoints and descriptors with SIFT
        kp1, des1 = sift.detectAndCompute(self.MarkImage[:, :, 0], None)
        kp2, des2 = sift.detectAndCompute(sceneImage[:, :, 0], None)

        # create BFMatcher object
        FLANN_INDEX_KDTREE = 0
        index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
        search_params = dict(checks=50)

        flann = cv2.FlannBasedMatcher(index_params, search_params)

        # Match descriptors.
        matches = flann.knnMatch(des1, des2, k=2)

        # Sort them in the order of their distance.
        good = []
        for m, n in matches:
            if m.distance < 0.7 * n.distance:
                good.append(m)
        if len(good) < self.MIN_MATCH_COUNT:
            return None

        src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        matchesMask = mask.ravel().tolist()

        h, w = self.MarkImage.shape[:2]
        pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
        dst = cv2.perspectiveTransform(pts, M)

        draw_params = dict(matchColor=(0, 255, 0),  # draw matches in green color
                           singlePointColor=None,
                           matchesMask=matchesMask,  # draw only inliers
                           flags=2)

        self.SceneImage = sceneImage
        self.DrawParams = draw_params
        self.KP1 = kp1
        self.KP2 = kp2
        self.GoodMatches = good
        return dst

    def getP(self, dst):
        """
        dst: 标记物关键点

        return self.MTX,self.DIST,self.RVEC,self.TVEC:
        反馈 内参、畸变系数，旋转向量，位移向量

        """
        if self.SceneImage is None:
            return None

        corners = np.float32([dst[1], dst[0], dst[2], dst[3]])
        gray = cv2.cvtColor(self.SceneImage, cv2.COLOR_BGR2GRAY)
        # 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), (1,0,0), (1,1,0)
        objp = np.zeros((2 * 2, 3), np.float32)
        objp[:, :2] = np.mgrid[0:2, 0:2].T.reshape(-1, 2)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

        if self.PTimes < self.PCount or self.PCount == 0:
            # Arrays to store object points and image points from all the images.
            objpoints = self.OBJPoints  # 3d point in real world space
            imgpoints = self.IMGPoints  # 2d points in image plane.

            if len(imgpoints) == 0 or np.sum(np.abs(imgpoints[-1] - corners2)) != 0:
                objpoints.append(objp)
                imgpoints.append(corners2)

            # Find mtx, dist, rvecs, tvecs
            ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
            if not ret:
                self.PTimes += 1
                return None
            self.OBJPoints = objpoints
            self.IMGPoints = imgpoints
            self.MTX = mtx
            self.DIST = dist
            self.RVEC = rvecs[0]
            self.TVEC = tvecs[0]
        else:
            # Find the rotation and translation vectors.
            _, rvec, tvec, _ = cv2.solvePnPRansac(objp, corners2, self.MTX, self.DIST)
            self.RVEC = rvec
            self.TVEC = tvec
        self.PTimes += 1

        return self.MTX, self.DIST, self.RVEC, self.TVEC

    def getGLP(self, width, height):
        """
        width, height: 场景宽、高

        """
        P = np.zeros(shape=(4, 4), dtype=np.float32)

        fx = self.MTX[0, 0]
        fy = self.MTX[1, 1]

        cx = self.MTX[0, -1]
        cy = self.MTX[1, -1]

        near = 0.1
        far = 100.0

        P[0, 0] = 2 * fx / width
        P[1, 1] = 2 * fy / height
        P[0, 2] = 1 - (2 * cx / width)
        P[1, 2] = (2 * cy / height) - 1
        P[2, 2] = -(far + near) / (far - near)
        P[3, 2] = -1.
        P[2, 3] = -(2 * far * near) / (far - near)

        p = P.T
        return p.flatten()

    def getGLM(self):
        R, _ = cv2.Rodrigues(self.RVEC)
        Rt = np.hstack((R, self.TVEC))

        Rx = np.array([[1, 0, 0], [0, -1, 0], [0, 0, -1]])

        M = np.eye(4)
        M[:3, :] = np.dot(Rx, Rt)

        m = M.T
        return m.flatten()

    # Debug code.
    def drawMatches(self, MarkImage, SceneImage):
        outImg = cv2.drawMatches(MarkImage, self.KP1,
                                 SceneImage, self.KP2,
                                 self.GoodMatches, None, **self.DrawParams)
        return outImg

    def drawBox(self, img):
        axis = np.float32([[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0],
                           [0, 0, -1], [0, 1, -1], [1, 1, -1], [1, 0, -1]])
        imgpts, jac = cv2.projectPoints(axis, self.RVEC, self.TVEC, self.MTX, self.DIST)
        imgpts = np.int32(imgpts).reshape(-1, 2)

        # draw pillars in blue color
        for i, j in zip(range(4), range(4, 8)):
            img2 = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]), (255, 0, 0), 3)

        # draw top layer in red color
        outImg = cv2.drawContours(img2, [imgpts[4:]], -1, (0, 0, 255), 3)

        return outImg


# Debug Code.
def debugMark():
    # Debug module.
    # from matplotlib import pyplot as plt

    markImage = cv2.imread('mark.png')
    sceneImage = cv2.imread('mark_in_scene.png')

    # Init PM.
    pm = GetPMatrix(markImage)

    # Get kp1, kp2, dst, goodMatches, [draw_params].
    dst = pm.findMark(sceneImage)
    if dst is None:
        exit()

    # # Get ret, mtx, dist, rvecs, tvecs
    tmp = None
    for i in range(30):
        tmp = pm.getP(dst)
        if tmp is None:
            exit()
        # print i
    mtx, dist, rvec, tvec = tmp

    # Draw Box
    h, w = markImage.shape[:2]
    sceneImage = pm.drawBox(sceneImage)

    # Draw corners.
    for point in dst:
        cv2.circle(sceneImage, tuple(point[0]), 5, (0, 0, 255), -1)

    h2, w2 = sceneImage.shape[:2]
    glP = pm.getGLP(w2, h2)
    glM = pm.getGLM()


    markImage = cv2.cvtColor(markImage, cv2.COLOR_BGR2RGB)
    sceneImage = cv2.cvtColor(sceneImage, cv2.COLOR_BGR2RGB)
    plt.figure('Mark test.')
    plt.subplot(121), plt.imshow(markImage), plt.title('Mark')
    plt.subplot(122), plt.imshow(sceneImage), plt.title('Scene')


def debugMatches():
    # Debug module.
    # from matplotlib import pyplot as plt

    markImage = cv2.imread('clock.png')
    sceneImage = cv2.imread('clock_in_scene.png')

    # Init PM.
    pm = GetPMatrix(markImage)

    # Get kp1, kp2, dst, goodMatches, [draw_params].
    dst = pm.getMatches(sceneImage)
    if dst is None:
        exit()

    # Draw circles and lines.
    img3 = pm.drawMatches(markImage, sceneImage)

    # # Get ret, mtx, dist, rvecs, tvecs
    tmp = None
    for i in range(30):
        tmp = pm.getP(dst)
        if tmp is None:
            exit()
        # print i
    mtx, dist, rvec, tvec = tmp

    # Draw Box
    h, w = markImage.shape[:2]
    img3[:, w:] = pm.drawBox(img3[:, w:])
    h2, w2 = sceneImage.shape[:2]
    glP = pm.getGLP(w2, h2)
    glM = pm.getGLM()

    print('mtx -------------')
    print(mtx)
    print('dist ------------')
    print(dist)
    print('rvec -----------')
    print(rvec)
    print('tvec -----------')
    print(tvec)
    print('glP ------------')
    print(glP)
    print('glM ------------')
    print(glM)

    img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
    plt.figure('Matches test.'), plt.imshow(img3)


if __name__ == '__main__':
    # Debug module.
    from matplotlib import pyplot as plt

    debugMark()
    debugMatches()
    plt.show()
```

- ar_demo

```
import cv2
import numpy as np

from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

import pygame, pygame.image
from pygame.locals import *

import BAR4Py.getPMatrix as getPMatrix
import BAR4Py.objloader as objloader


def getGLPM(markImage, sceneImage, isMatches):
    height, width = sceneImage.shape[:2]

    # Init PM.
    pm = getPMatrix.GetPMatrix(markImage)
    # Get dst.
    dst = None
    if isMatches:
        dst = pm.getMatches(sceneImage)
    else:
        dst = pm.findMark(sceneImage)
    if dst is None:
        exit()
    # print dst
    # Get ret, mtx, dist, rvecs, tvecs
    tmp = pm.getP(dst)
    if tmp is None:
        exit()
    mtx, _, rvec, tvec = tmp
    # Debug code.
    # print 'mtx:\n',mtx,'\nrvec:\n',rvec,'\ntvec:\n',tvec

    glP = pm.getGLP(width, height)
    glM = pm.getGLM()
    # Debug code.
    # print 'glP:\n',glP,'\nglM:\n',glM

    return glP, glM


def set_projection_from_camera(glP):
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()

    glLoadMatrixf(glP)


def set_modelview_from_camera(glM, scale=1.):
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    glLoadMatrixf(glM)
    glTranslate(0.5, 0.5, -0.5)
    glRotate(180, 1, 0, 0)
    glRotate(180, 0, 0, 1)
    glScalef(scale, scale, scale)


def draw_background(imgName):
    bg_image = pygame.image.load(imgName).convert()
    bg_data = pygame.image.tostring(bg_image, 'RGBX', 1)
    width, height = bg_image.get_size()

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glEnable(GL_TEXTURE_2D)
    glGT = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, glGT)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, bg_data)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)

    glBegin(GL_QUADS)
    glTexCoord2f(0.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0)
    glTexCoord2f(1.0, 0.0);
    glVertex3f(1.0, -1.0, -1.0)
    glTexCoord2f(1.0, 1.0);
    glVertex3f(1.0, 1.0, -1.0)
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0)
    glEnd()

    glDeleteTextures(1)

    return glGT


def load_and_draw_model(filName):
    glLightfv(GL_LIGHT0, GL_POSITION, (-50, 200, 250, 0.0))
    glLightfv(GL_LIGHT0, GL_AMBIENT, (0.2, 0.2, 0.2, 1.0))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (0.5, 0.5, 0.5, 1.0))

    glEnable(GL_LIGHT0)
    glEnable(GL_LIGHTING)
    glEnable(GL_COLOR_MATERIAL)
    glEnable(GL_DEPTH_TEST)
    glShadeModel(GL_SMOOTH)

    obj = objloader.OBJ(filName, swapyz=True)
    glCallList(obj.gl_list)

    return obj


class BAR4Py:
    def __init__(self, captionStr, markImageName, sceneImageName, OBJFileName, isMatches=False):
        markImage = cv2.imread(markImageName)
        sceneImage = cv2.imread(sceneImageName)
        height, width = sceneImage.shape[:2]

        # Init pygame.
        pygame.init()
        pygame.display.set_mode((width, height), OPENGL | DOUBLEBUF)
        pygame.display.set_caption(captionStr)

        draw_background(sceneImageName)

        glP, glM = getGLPM(markImage, sceneImage, isMatches)
        set_projection_from_camera(glP)
        if isMatches:
            set_modelview_from_camera(glM, 0.5)
        else:
            set_modelview_from_camera(glM)

        load_and_draw_model(OBJFileName)

    def run(self):
        while True:
            event = pygame.event.poll()
            if event.type in (QUIT, KEYDOWN):
                break

            pygame.display.flip()


if __name__ == '__main__':
    import sys

    if len(sys.argv) == 2:
        if sys.argv[1] == 'mark':
            bar4py = BAR4Py('BAR4Py Demo.', './mark.png', './mark_in_scene.png', './hj.obj')
        elif sys.argv[1] == 'matches':
            bar4py = BAR4Py('BAR4Py Demo.', './clock.png', './clock_in_scene.png', './hj.obj', True)
        bar4py.run()
```

- 运行代码


BAR4Py Demo.