**การปรับเทียบกล้อง**

เป้าหมาย

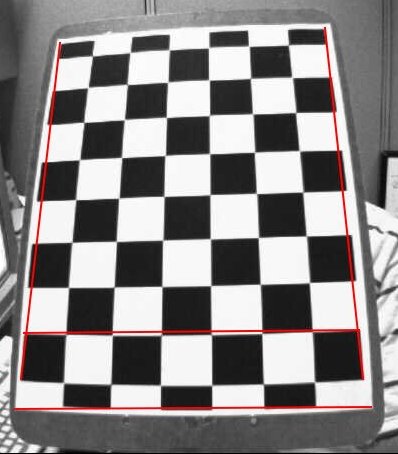
ในส่วนนี้,

* เราจะเรียนรู้เกี่ยวกับการบิดเบือนในกล้องพารามิเตอร์ภายในและภายนอกของกล้อง ฯลฯ
* เราจะเรียนรู้ที่จะหาพารามิเตอร์เหล่านี้ undistort ภาพ ฯลฯ

พื้นฐาน

กล้องรูรับแสงราคาถูกในปัจจุบันนำเสนอภาพที่ผิดเพี้ยนมาก การบิดเบือนที่สำคัญสองประการคือการบิดเบี้ยวในแนวรัศมีและการบิดเบือนการสัมผัส

เนื่องจากการบิดเบี้ยวในแนวรัศมีเส้นตรงจะปรากฏโค้ง ผลกระทบของมันมากขึ้นขณะที่เราเคลื่อนห่างจากจุดกึ่งกลางของภาพ ตัวอย่างเช่นภาพหนึ่งภาพแสดงด้านล่างโดยที่ขอบทั้งสองด้านของกระดานหมากรุกมีเส้นสีแดงอยู่ แต่คุณจะเห็นว่าเส้นขอบไม่ได้เป็นเส้นตรงและไม่ตรงกับเส้นสีแดง ทุกเส้นตรงที่คาดไว้จะพองออก ไปที่[Distortion (optics)](http://en.wikipedia.org/wiki/Distortion_%28optics%29)เพื่อดูรายละเอียดเพิ่มเติม



การบิดเบือนนี้จะแก้ไขได้ดังนี้:

x_ {corrected} = x (1 + k_1 r ^ 2 + k_2 r ^ 4 + k_3 r ^ 6) \\ y_ {แก้ไข} = y (1 + k_1 r ^ 2 + k_2 r ^ 4 + k_3 r ^ 6)

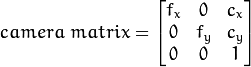
ในทำนองเดียวกันการบิดเบือนอื่นคือการบิดเบี้ยวที่บิดเบี้ยวซึ่งเกิดขึ้นเนื่องจากการจับภาพเลนส์ไม่สอดคล้องกันอย่างสมบูรณ์แบบขนานกับระนาบการถ่ายภาพ ดังนั้นพื้นที่บางส่วนของภาพอาจดูใกล้กว่าที่คาดไว้ จะแก้ไขได้ดังนี้:

x_ {corrected} = x + [2p_1xy + p_2 (r ^ 2 + 2x ^ 2)] \\ y_ {แก้ไข} = y + [p_1 (r ^ 2 + 2y ^ 2) + 2p_2xy]

ในระยะสั้นเราต้องหาพารามิเตอร์ห้าตัวซึ่งเรียกว่าสัมประสิทธิ์การบิดเบือนโดย:

การบิดเบือน \;  coefficients = (k_1 \ hspace {10pt} k_2 \ hspace {10pt} p_1 \ hspace {10pt} p_2 \ hspace {10pt} k_3)

นอกเหนือจากนี้เราจำเป็นต้องค้นหาข้อมูลเพิ่มเติมเช่นพารามิเตอร์ภายในและภายนอกของกล้อง พารามิเตอร์เฉพาะที่เจาะจงสำหรับกล้อง ประกอบด้วยข้อมูลเช่นความยาวโฟกัส ( f_x, f_y), ศูนย์ออปติคอล ( c_x, c_y) เป็นต้นเรียกว่าเมทริกซ์กล้อง ขึ้นอยู่กับกล้องเท่านั้นดังนั้นการคำนวณครั้งเดียวจึงสามารถจัดเก็บไว้เพื่อวัตถุประสงค์ในอนาคต มันแสดงเป็นเมทริกซ์ 3x3:



พารามิเตอร์ภายนอกหมายถึงการหมุนและเวกเตอร์การแปลซึ่งแปลพิกัดของจุด 3D เป็นระบบพิกัด

สำหรับแอพพลิเคชันสเตอริโอการบิดเบือนเหล่านี้จำเป็นต้องได้รับการแก้ไขก่อน เพื่อหาพารามิเตอร์เหล่านี้ทั้งหมดสิ่งที่เราต้องทำคือการจัดเตรียมภาพตัวอย่างบางส่วนของรูปแบบที่กำหนดไว้อย่างชัดเจน (เช่นกระดานหมากรุก) เราหาจุดที่เฉพาะเจาะจงบางอย่างในนั้น (มุมสี่เหลี่ยมในกระดานหมากรุก) เรารู้พิกัดของมันในพื้นที่โลกแห่งความจริงและเรารู้พิกัดของมันในภาพ ด้วยข้อมูลเหล่านี้ปัญหาทางคณิตศาสตร์บางอย่างจะได้รับการแก้ไขในพื้นหลังเพื่อให้ได้สัมประสิทธิ์การบิดเบือน นั่นคือบทสรุปของเรื่องราวทั้งหมด เพื่อให้ได้ผลลัพธ์ที่ดีขึ้นเราจำเป็นต้องมีรูปแบบการทดสอบอย่างน้อย 10 แบบ

รหัส

ดังที่ได้กล่าวมาแล้วเราจำเป็นต้องมีรูปแบบการทดสอบ 10 ขั้นต่ำสำหรับการปรับเทียบกล้อง OpenCV มาพร้อมกับภาพหมากรุกบางแผ่น (ดูตัวอย่าง / cpp / left01.jpg - left14.jpg ) ดังนั้นเราจะใช้ประโยชน์จากมัน เพื่อความเข้าใจให้พิจารณาเพียงภาพเดียวของกระดานหมากรุก ข้อมูลข้อมูลสำคัญที่จำเป็นสำหรับการปรับเทียบกล้องคือชุดของจุดโลกแห่งความเป็นจริง 3 มิติและจุดภาพ 2D ที่ตรงกัน จุดภาพ 2D เป็นสิ่งที่เราสามารถหาได้จากภาพ (จุดภาพเหล่านี้คือตำแหน่งที่สี่เหลี่ยมสีดำสี่เหลี่ยมสัมผัสกันในกระดานหมากรุก)

สิ่งที่เกี่ยวกับจุด 3D จากพื้นที่โลกแห่งความจริง? ภาพเหล่านี้ถ่ายจากกล้องถ่ายภาพนิ่งและกระดานหมากรุกวางอยู่ในตำแหน่งต่างๆและทิศทาง ดังนั้นเราจำเป็นต้องรู้(X, Y, Z)ค่า แต่สำหรับความเรียบง่ายเราสามารถพูดได้ว่ากระดานหมากรุกถูกเก็บไว้นิ่งที่ระนาบ XY (ดังนั้น Z = 0 เสมอ) และกล้องถูกย้ายไปตามลำดับ การพิจารณานี้ช่วยให้เราสามารถหาค่า X, Y เท่านั้น ตอนนี้สำหรับค่า X, Y เราสามารถผ่านจุดเป็น (0,0), (1,0), (2,0), ... ซึ่งหมายถึงตำแหน่งของจุด ในกรณีนี้ผลลัพธ์ที่เราได้รับจะอยู่ในขนาดของกระดานสี่เหลี่ยมจัตุรัส แต่ถ้าเรารู้ขนาดของตาราง (พูด 30 มิลลิเมตร) และเราสามารถส่งผ่านค่าเป็น (0,0), (30,0), (60,0), ... , เราได้ผลลัพธ์เป็นมิลลิเมตร (ในกรณีนี้เราไม่ทราบขนาดรูปสี่เหลี่ยมจัตุรัสเนื่องจากเราไม่ได้ถ่ายภาพเหล่านี้ดังนั้นเราจึงถือว่าเป็นรูปสี่เหลี่ยมจัตุรัส)

จุด 3D เรียกว่า**จุดวัตถุ**และจุดภาพ 2D เรียกว่า**จุดภาพ**

ตั้งค่า

ดังนั้นเพื่อหารูปแบบในกระดานหมากรุกเราจะใช้ฟังก์ชั่นcv2.findChessboardCorners () เรายังต้องผ่านรูปแบบที่เรากำลังมองหาเช่น 8x8 grid, 5x5 grid เป็นต้นในตัวอย่างนี้เราใช้ตาราง 7x6 (โดยปกติกระดานหมากรุกมีช่อง 8x8 และมุมภายใน 7x7) มันส่งกลับจุดมุมและ retval ซึ่งจะเป็นจริงถ้าได้รับรูปแบบ มุมเหล่านี้จะถูกจัดเรียงตามลำดับ (จากซ้ายไปขวาบนสุดไปด้านล่าง)

**ดูสิ่งนี้ด้วย :** ฟังก์ชั่นนี้อาจไม่สามารถหารูปแบบที่ต้องการในภาพทั้งหมดได้ ดังนั้นหนึ่งในตัวเลือกที่ดีคือการเขียนโค้ดดังกล่าวจะเริ่มต้นกล้องและตรวจสอบแต่ละเฟรมสำหรับรูปแบบที่ต้องการ เมื่อได้รูปแบบแล้วให้หามุมและเก็บไว้ในรายการ นอกจากนี้ยังมีช่วงเวลาก่อนที่จะอ่านเฟรมถัดไปเพื่อให้เราสามารถปรับกระดานหมากรุกของเราในทิศทางที่ต่างกันได้ ดำเนินขั้นตอนต่อไปจนกว่าจะได้รับรูปแบบที่ดีจำนวนที่ต้องการ แม้ในตัวอย่างที่ให้ไว้ที่นี่เราไม่แน่ใจจาก 14 ภาพที่ระบุว่ามีกี่สิ่งที่ดี ดังนั้นเราจึงอ่านภาพทั้งหมดและถ่ายรูปที่ดี

**ดูสิ่งนี้ด้วย :** แทนบอร์ดหมากรุกเราสามารถใช้ตารางวงกลมบางส่วนได้ แต่ใช้ฟังก์ชัน**cv2.findCirclesGrid ()**เพื่อหารูปแบบ มีคนบอกว่าจำนวนภาพน้อยพอจะใช้ตารางวงกลมได้

เมื่อเราหามุมที่เราสามารถเพิ่มความถูกต้องของตนโดยใช้**cv2.cornerSubPix**() นอกจากนี้เรายังสามารถวาดลวดลายโดยใช้**cv2.drawChessboardCorners**() ขั้นตอนทั้งหมดนี้รวมอยู่ในโค้ดด้านล่าง:

**import** **numpy** **as** **np**

**import** **cv2**

**import** **glob**

*# 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.*

images = glob.glob('\*.jpg')

**for** fname **in** images:

img = cv2.imread(fname)

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

*# Find the chess board corners*

ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

*# If found, add object points, image points (after refining them)*

**if** ret == True:

objpoints.append(objp)

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

imgpoints.append(corners)

*# Draw and display the corners*

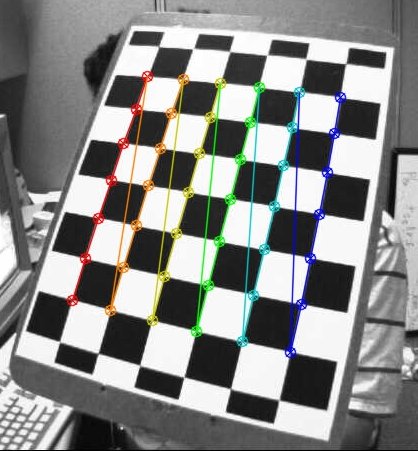
cv2.drawChessboardCorners(img, (7,6), corners2,ret)

cv2.imshow('img',img)

cv2.waitKey(500)

cv2.destroyAllWindows()

**หนึ่งภาพที่มีรูปแบบที่วาดบนจะแสดงด้านล่าง:**



การสอบเทียบ

ดังนั้นตอนนี้เรามีจุดออบเจกต์และจุดภาพที่เราพร้อมที่จะไปปรับเทียบ เพื่อที่เราจะใช้ฟังก์ชั่น**cv2.calibrateCamera**() จะส่งกลับเมทริกซ์กล้องค่าสัมประสิทธิ์การบิดเบือนการหมุนและเวกเตอร์การแปลเป็นต้น

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

ยกเลิกการจัดเรียง

เรามีสิ่งที่เราพยายาม ตอนนี้เราสามารถถ่ายภาพและล้างข้อมูลได้ OpenCV มาพร้อมกับสองวิธีเราจะเห็นทั้งสองอย่าง แต่ก่อนที่เราสามารถปรับแต่งเมทริกซ์กล้องอยู่บนพื้นฐานของการปรับพารามิเตอร์ฟรีโดยใช้cv2.getOptimalNewCameraMatrix () หากพารามิเตอร์การปรับค่าalpha = 0จะแสดงภาพที่ไม่ถูกบิดเบือนด้วยพิกเซลที่ไม่ต้องการขั้นต่ำ ดังนั้นจึงอาจลบพิกเซลบางส่วนที่มุมภาพ หากalpha = 1พิกเซลทั้งหมดจะถูกเก็บไว้กับภาพสีดำพิเศษบางอย่าง นอกจากนี้ยังส่งกลับ ROI ของภาพซึ่งสามารถใช้ในการครอบตัดผล

ดังนั้นเราจึงใช้ภาพใหม่ ( left12.jpgในกรณีนี้นั่นคือภาพแรกในบทนี้)

img = cv2.imread('left12.jpg')

h, w = img.shape[:2]

newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))

1. ใช้**cv2.undistort ()**

นี่เป็นเส้นทางที่สั้นที่สุด เพียงเรียกใช้ฟังก์ชันและใช้ ROI ที่ได้รับข้างต้นเพื่อตัดผลลัพธ์

*# undistort*

dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

*# crop the image*

x,y,w,h = roi

dst = dst[y:y+h, x:x+w]

cv2.imwrite('calibresult.png',dst)

2. ใช้**remapping**

นี่เป็นเส้นทางโค้ง ขั้นแรกหาฟังก์ชันการทำแผนที่จากภาพที่บิดเบี้ยวไปยังภาพที่ไม่ถูกบิดเบือน จากนั้นใช้ฟังก์ชัน remap

*# undistort*

mapx,mapy = cv2.initUndistortRectifyMap(mtx,dist,None,newcameramtx,(w,h),5)

dst = cv2.remap(img,mapx,mapy,cv2.INTER\_LINEAR)

*# crop the image*

x,y,w,h = roi

dst = dst[y:y+h, x:x+w]

cv2.imwrite('calibresult.png',dst)

ทั้งสองวิธีให้ผลลัพธ์เช่นเดียวกัน ดูผลลัพธ์ด้านล่าง:



คุณสามารถเห็นในผลลัพธ์ที่ขอบทั้งหมดจะตรง

ตอนนี้คุณสามารถเก็บเมทริกซ์ของกล้องและค่าสัมประสิทธิ์การบิดเบือนโดยใช้ฟังก์ชันการเขียนใน Numpy (np.savez, np.savetxt ฯลฯ ) เพื่อใช้ในอนาคต

Re-projection Error

ข้อผิดพลาดในการฉายใหม่ให้ค่าประมาณที่ดีเพียงว่าพารามิเตอร์ที่พบมีค่าเท่าไร นี่ควรใกล้เคียงกับศูนย์มากที่สุด เราได้เปลี่ยนจุดเริ่มต้นของวัตถุไปยังจุดภาพโดยใช้cv2.projectPoints ()แล้วแปลงรูปที่อยู่ภายในการบิดเบือนการหมุนและการแปล จากนั้นเราคำนวณบรรทัดฐานสัมบูรณ์ระหว่างสิ่งที่เราได้รับกับการเปลี่ยนแปลงของเราและขั้นตอนการหามุม เพื่อหาข้อผิดพลาดโดยเฉลี่ยเราจะคำนวณค่าเฉลี่ยเลขคณิตของข้อผิดพลาดที่คำนวณได้สำหรับภาพที่ปรับเทียบทั้งหมด

mean\_error = 0

**for** i **in** xrange(len(objpoints)):

imgpoints2, \_ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)

error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM\_L2)/len(imgpoints2)

tot\_error += error

**print** "total error: ", mean\_error/len(objpoints)