

3D Reconstruction with Stereo Images - Part 2: Pose Estimation

AI/HUB

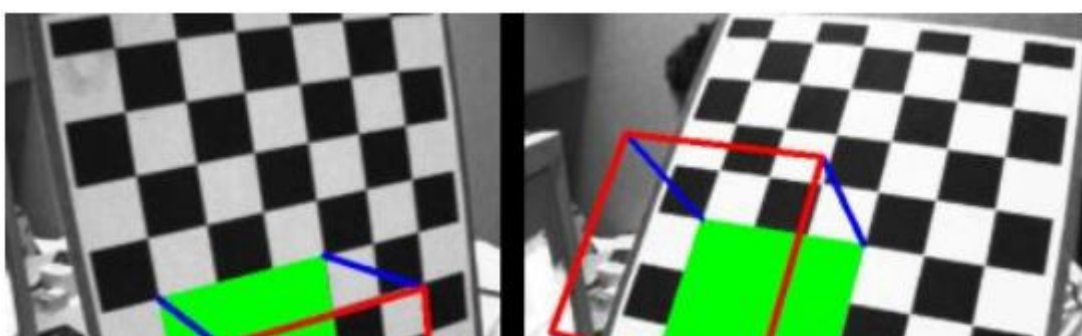
AI/HUB

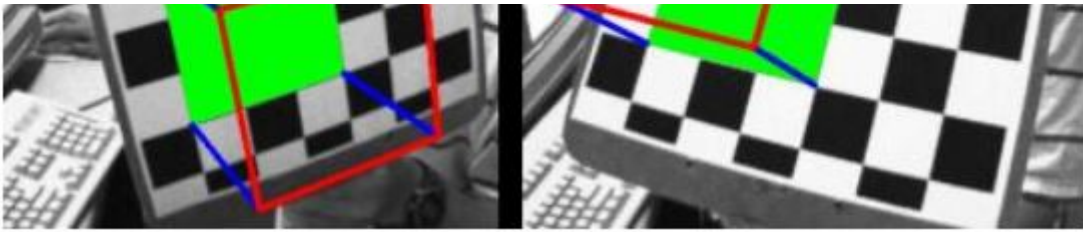
Nov 7, 2018 · 5 min read



Introduction

In our previous post we worked towards understanding the features of our cameras and how that information relates to our goal of full 3D reconstruction. Through calibration we now have important figures that describe the characteristics of our cameras that are usable by our program, that is; the camera matrix and distortion coefficients. From this information we can take a picture of a patterned image with our camera and calculate how that object is situated in real world space. For our example; we will use the chessboard image and visualize the planar objects relative position by drawing a 3D cube aligned to its orientation.





Getting Started

The goal of this exercise will be to draw an X, Y, Z axis onto our image, positioned at the chessboard's bottom corner. When working in 3D space it is typical to denote the X axis in blue, Y axis in green, and the Z axis in red. For this case we will set the Z axis as perpendicular to the patterned object (the axis that points away from the 2D plane of the chess board and toward the camera).

Let's start by retrieving our saved camera matrix, and distortion coefficient from the past exercise.

```
import cv2
import numpy as np
import glob

# Load previously saved data
with np.load('B.npz') as X:
    mtx, dist, _, _ = [X[i] for i in ('mtx', 'dist', 'rvecs', 'tvecs')]
```

Next is to write a function that will draw a 3D axis over the image. We will need to feed this function; the chessboard image, the 4 corners of the chessboard, and 3 points that represent the 'end point' of our axis lines in each 3 directions.

The corners of the chessboard can be found with the previously used `cv2.findChessboardCorner()` function. It will return an array with the position of the 4 corners needed. For our case we only need to focus on the bottom left hand corner, which is first element of the array. See below (`corner[0]`). Once this corner is retrieved we can then extend our axis from that point using the 3 end points we defined earlier.

```
def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv2.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0),
5)
    img = cv2.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0),
5)
    img = cv2.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255),
```

```
5)
    return img
```

Our next step is to identify the points in 3D space that will be used to help draw our axis. Remember from our last exercise we set one unit to equal the length of a single square on the chessboard. For this example we will create an axis that spans the length of 3 units, or chessboard squares, in each direction. For the Z axis we must denote a negative value which will ensure the axis line drawn will face the camera.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)
objp = np.zeros((6*7,3), np.float32)
objp[:, :2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
```

At this point we can employ our draw function. First step is to load the image, search for the 7 x 6 grid. If it is found we can calculate its rotation and translation using the function `cv2.solvePnPRansac()`. Now we project the axis points from points in 3D space to points on the 2D image plane. When the axis is identified we can then use the draw function to visualize its orientation.

```
for fname in glob.glob('left*.jpg'):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

    if ret == True:
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),
(-1,-1),criteria)

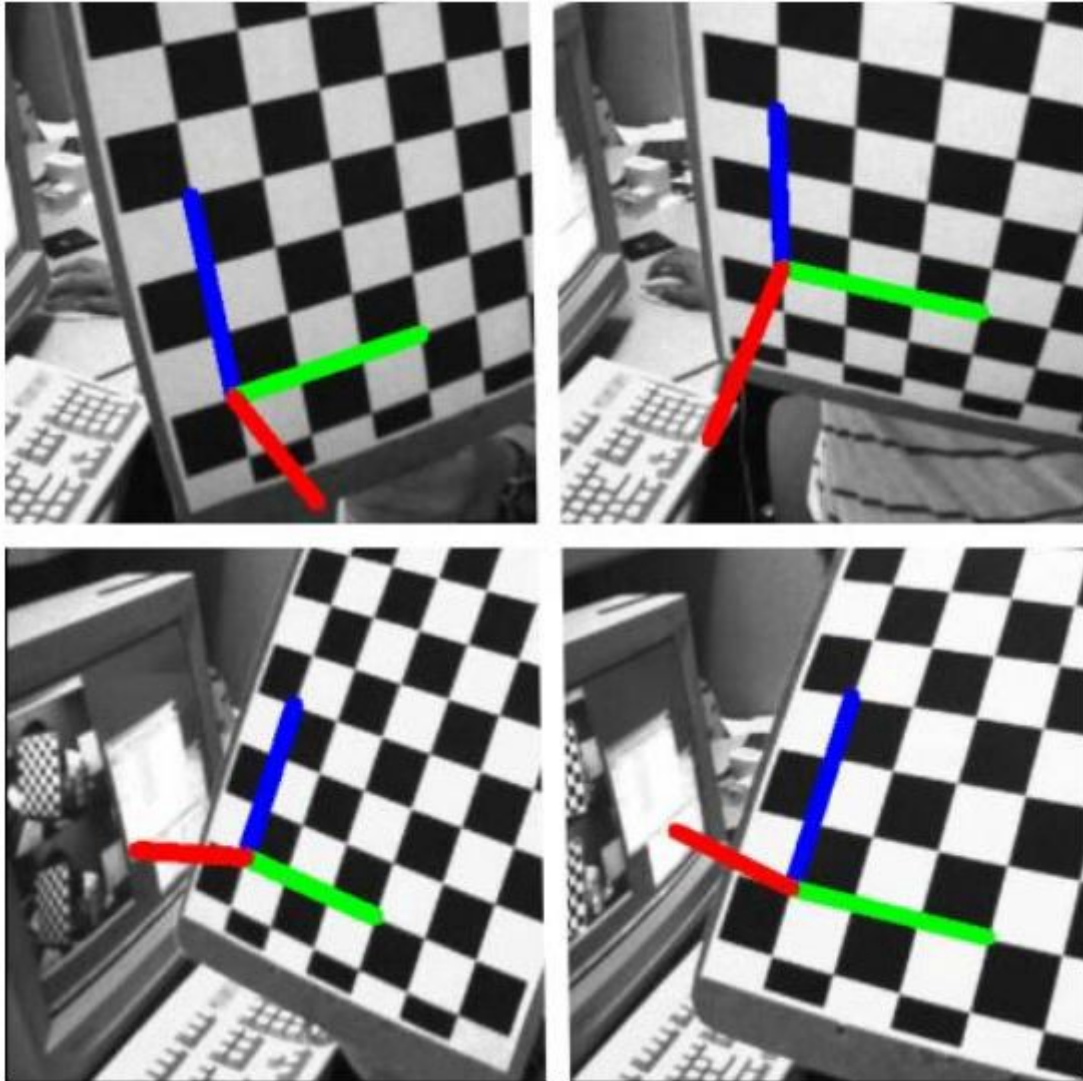
        # Find the rotation and translation vectors.
        rvecs, tvecs, inliers = cv2.solvePnPRansac(objp, corners2,
mtx, dist)

        # project 3D points to image plane
        imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx,
dist)

        img = draw(img,corners2,imgpts)
        cv2.imshow('img',img)
        k = cv2.waitKey(0) & 0xff
        if k == 's':
            cv2.imwrite(fname[:6]+' .png', img)
```

```
cv2.destroyAllWindows()
```

Below we can see our program in action over 4 different images of a chess board.



Render a cube

In order to draw a cube we can modify the draw function as follows. A 3 x 3 square will be drawn first, oriented to the bottom left corner and parallel to our chessboard. Then we will add lines extending from this square to complete the cube in the direction that faces the camera.

```
def draw(img, corners, imgpts):
    imgpts = np.int32(imgpts).reshape(-1,2)

    # draw ground floor in green
    img = cv2.drawContours(img, [imgpts[:4]], -1, (0,255,0), -3)

    # draw pillars in blue color
```



```

for i,j in zip(range(4),range(4,8)):
    img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]),
(255),3)

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

return img

```

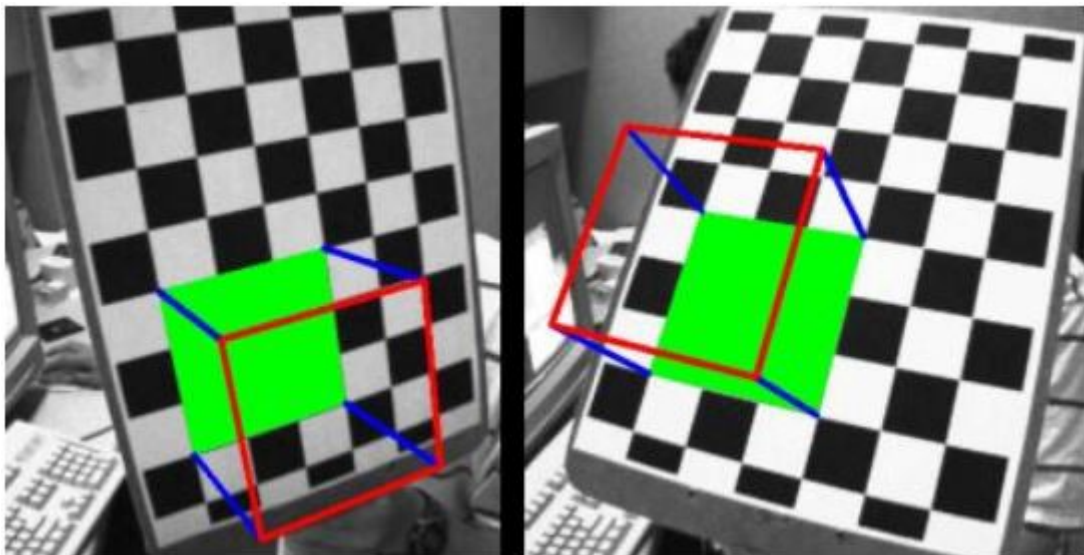
The axis points must be modified to include each corner of the cube.

```

axis = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
[0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3] ])

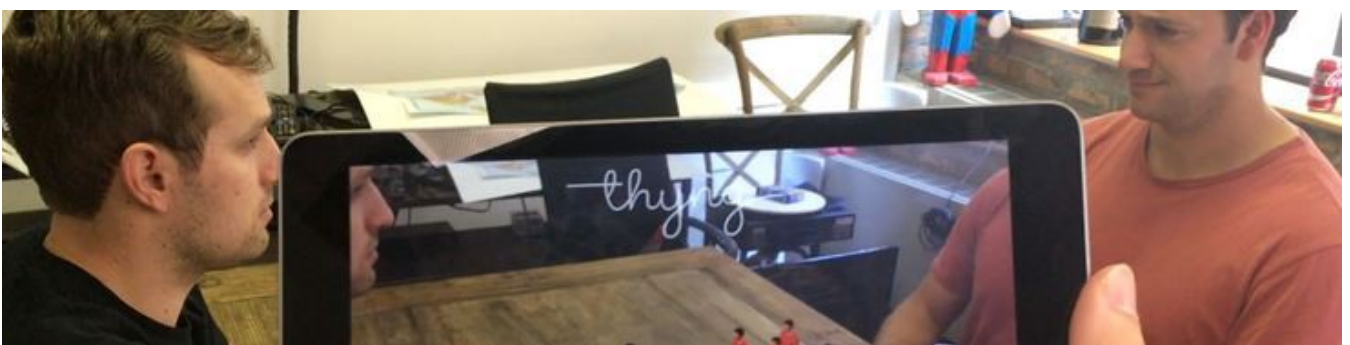
```

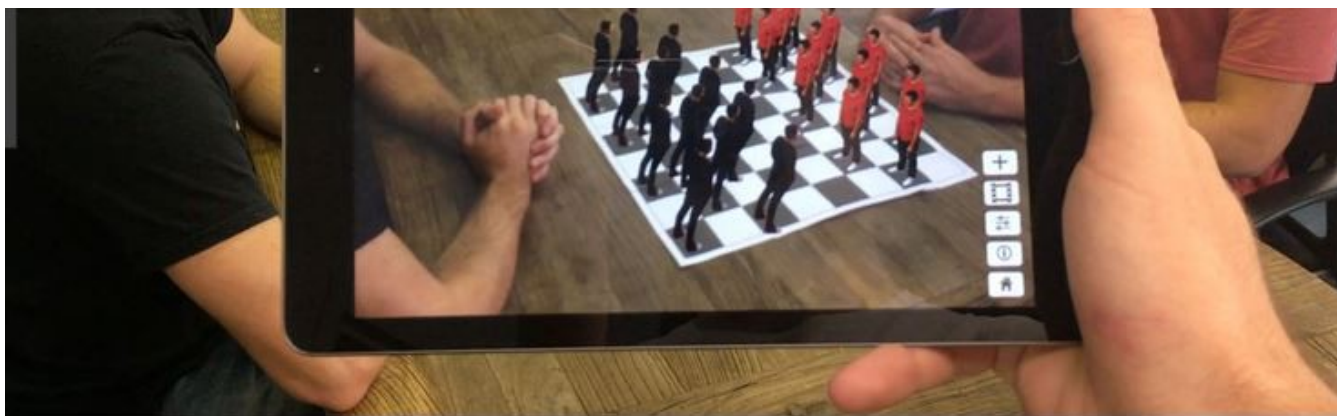
The results are shown in two images below.



Conclusion

After the above exercise consider the applications for augmented reality. Here we have successfully included a 3D object in a real world image that interacts with the object that is pictured, allowing for a dynamic link between the real and virtual worlds.







Written by: Keenan James under supervision of Professor Amit Maraj

Machine Learning AI Opencv Computer Vision

[About](#) [Help](#) [Legal](#)

Get the Medium app

 A button that says 'Download on the App Store', and if clicked it will lead you to the iOS App store

 A button that says 'Get it on, Google Play', and if clicked it will lead you to the Google Play store