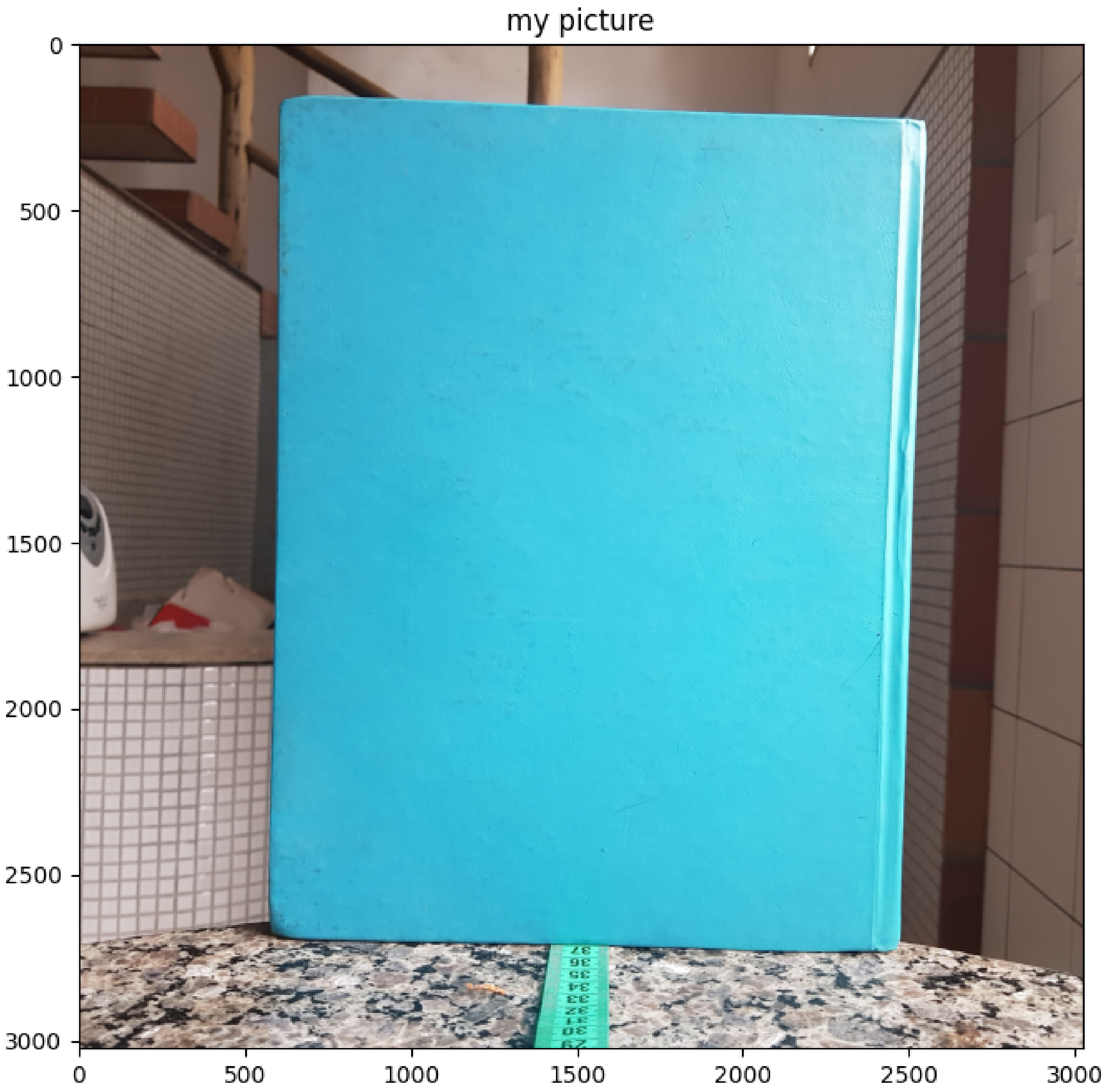
In [29]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import glob
import sys
import os

plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['figure.dpi'] = 100 # 200 e.g. is really fine, but slower
```

Naive

In [30]:

```
image = cv2.imread("Livro.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(gray)
plt.title('my picture')
plt.show()
```



The image above have:

```
width=3021px
heigth=3021px
```

The book in the image have:

```
width=1938px
heigth=2542px
```

The real size of the book is:

```
width=22.85cm
height=28.4cm
```

And its center is located about 37cm from the camera position.

That being, we can aproximate:

```
fx = 37*(1938/22.85)
```

In [31]:

```
fx = 37*(1938/22.85)
fx
```

Out[31]: 3138.1181619256013

In [32]:

```
fy = 37*(2542/28.4)
fy
```

Out[32]: 3311.760563380282

In [33]:

```
cx = 3021/2
cx
```

Out[33]: 1510.5

In [34]:

```
cy = 3021/2
cy
```

Out[34]: 1510.5

Octave

Using the Octave Calibration Toolbox (https://github.com/nghiaho12/camera_calibration_toolbox_octave) we obtain:

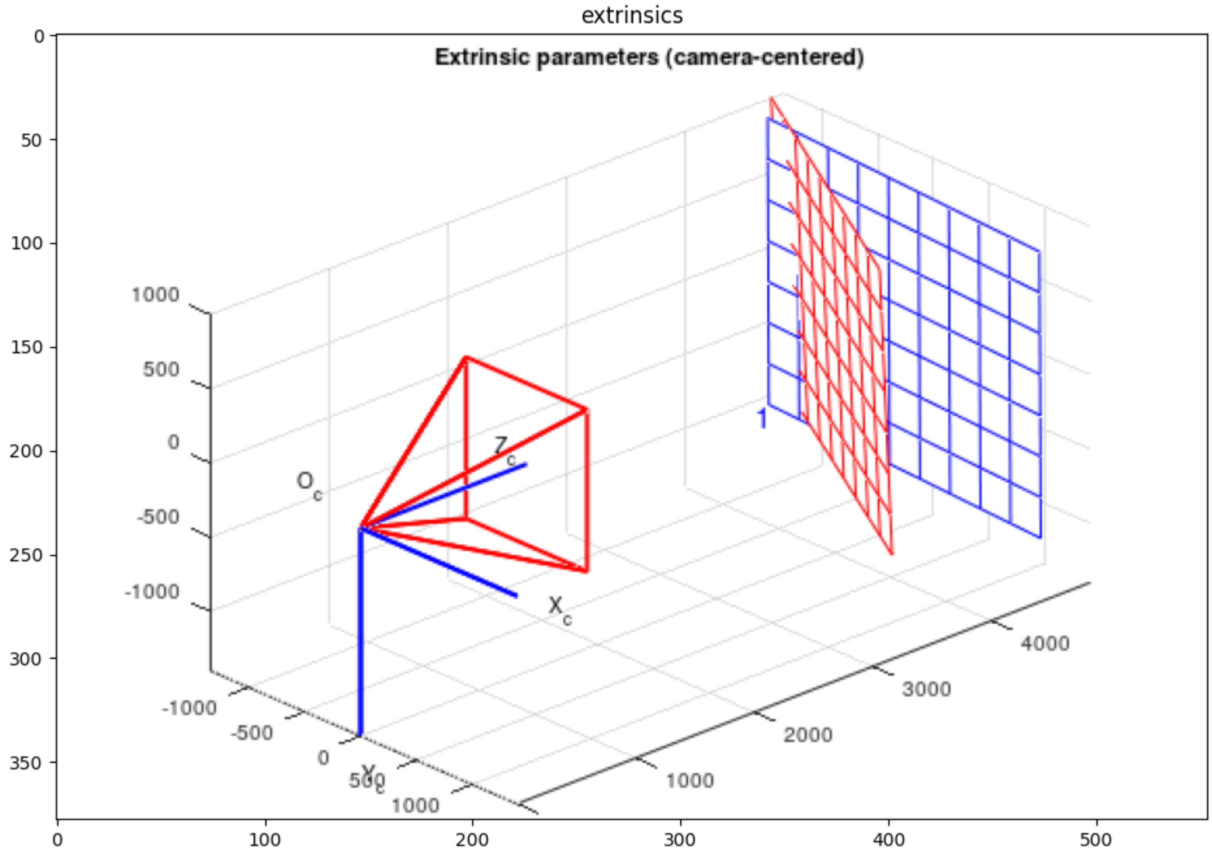
In [35]:

```
octave1 = cv2.imread("octave1.png")
octave2 = cv2.imread("octave2.png")
octave3 = cv2.imread("octave3.png")
octave4 = cv2.imread("octave4.png")
octave5 = cv2.imread("octave5.png")
```

```
octave1_d = cv2.cvtColor(octave1, cv2.COLOR_BGR2RGB)
octave2_d = cv2.cvtColor(octave2, cv2.COLOR_BGR2RGB)
octave3_d = cv2.cvtColor(octave3, cv2.COLOR_BGR2RGB)
octave4_d = cv2.cvtColor(octave4, cv2.COLOR_BGR2RGB)
octave5_d = cv2.cvtColor(octave5, cv2.COLOR_BGR2RGB)
```

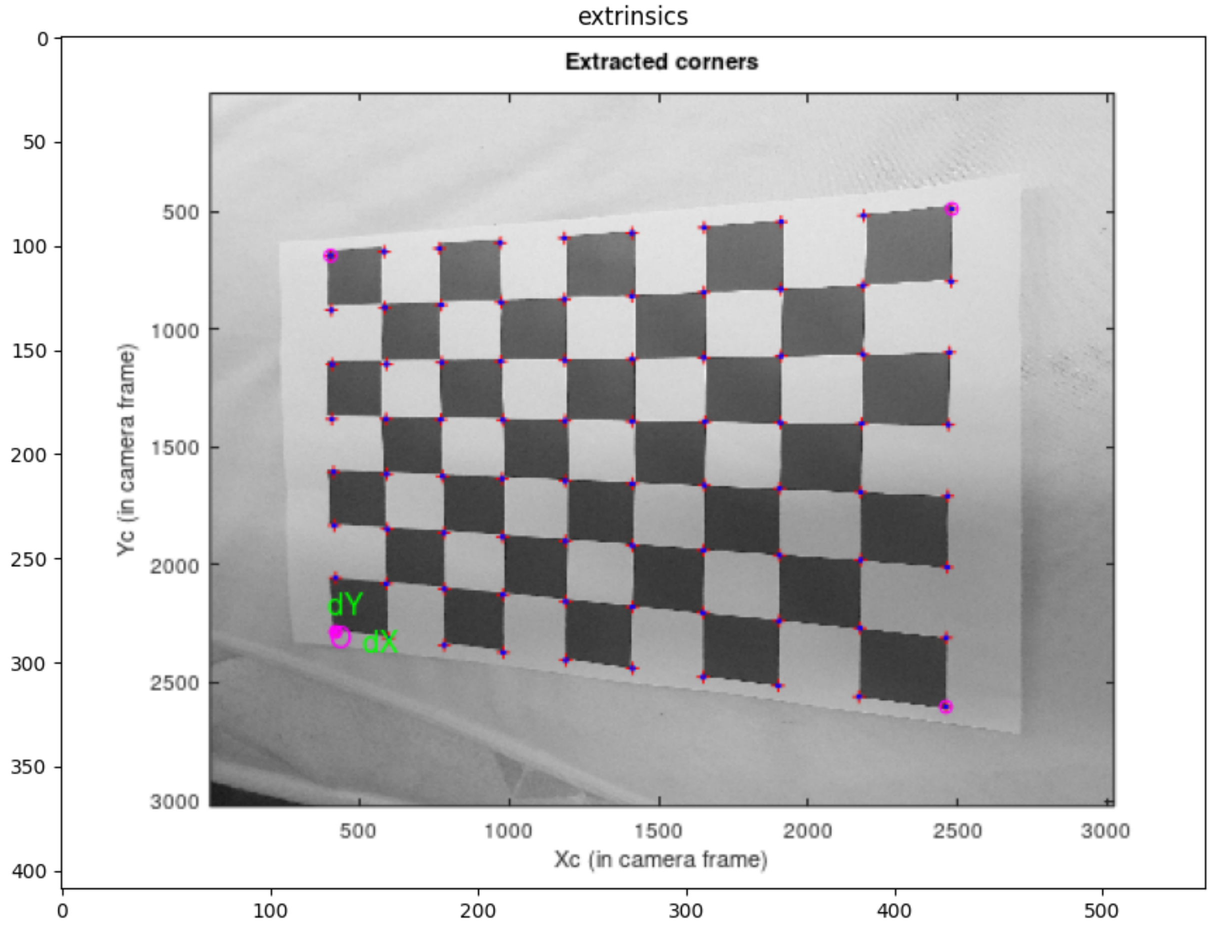
In [36]:

```
plt.imshow(octave1_d)
plt.title('extrinsics')
plt.show()
```

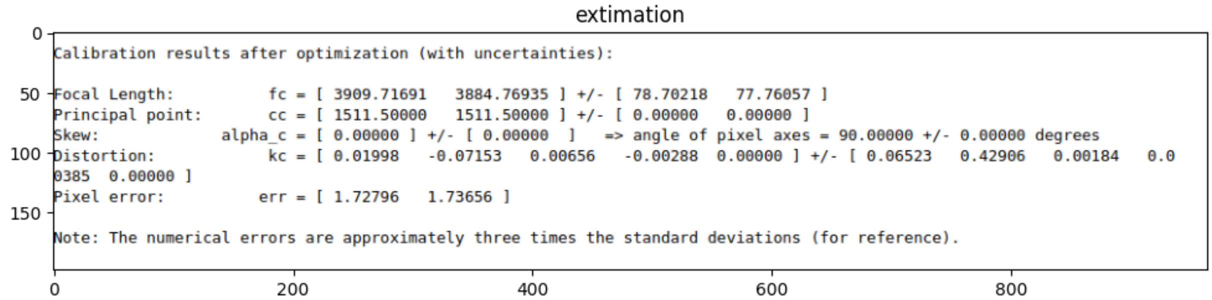


In [37]:

```
plt.imshow(octave2_d)
plt.title('extrinsics')
plt.show()
```



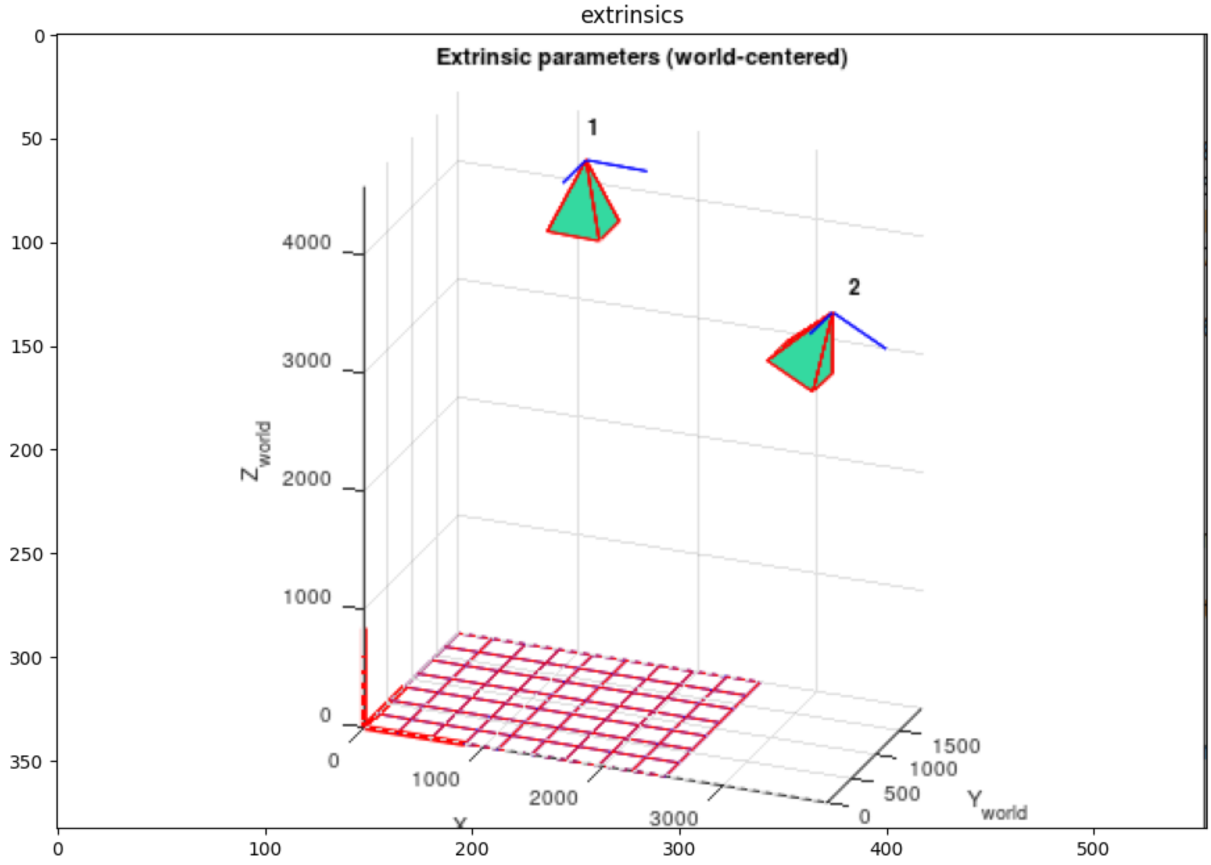
In [38]:
```python
plt.imshow(octave3_d)
plt.title('extimation')
plt.figure(figsize=(20, 20))
plt.show()
```


extimation

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 3909.71691   3884.76935 ] +/- [ 78.70218   77.76057 ]
Principal point:       cc = [ 1511.50000   1511.50000 ] +/- [ 0.00000   0.00000 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.01998   -0.07153   0.00656   -0.00288  0.00000 ] +/- [ 0.06523   0.42906   0.00184   0.0
0385  0.00000 ]
Pixel error:          err = [ 1.72796   1.73656 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

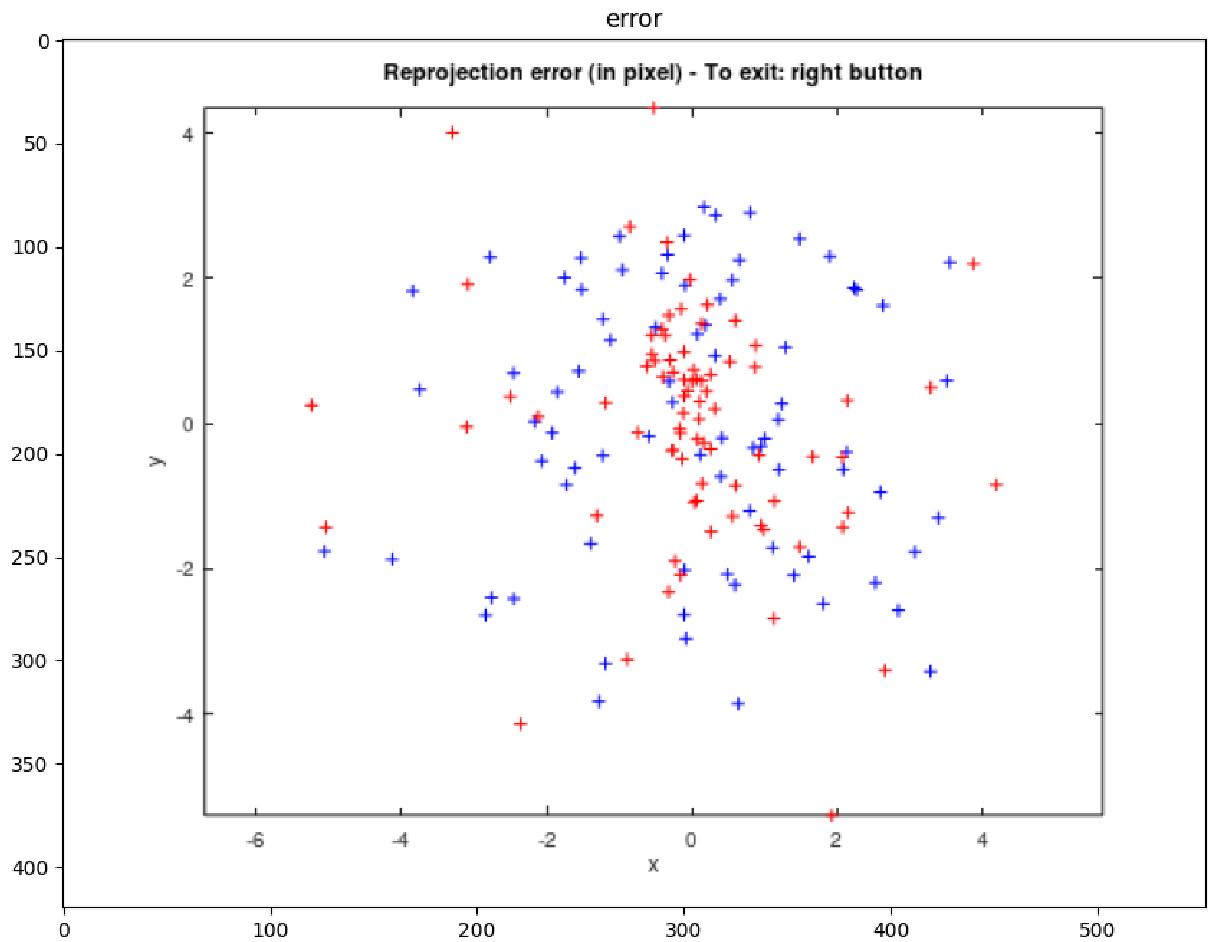
<Figure size 2000x2000 with 0 Axes>

In [39]:
```python
plt.imshow(octave4_d)
plt.title('extrinsics')
plt.show()
```



In [40]:

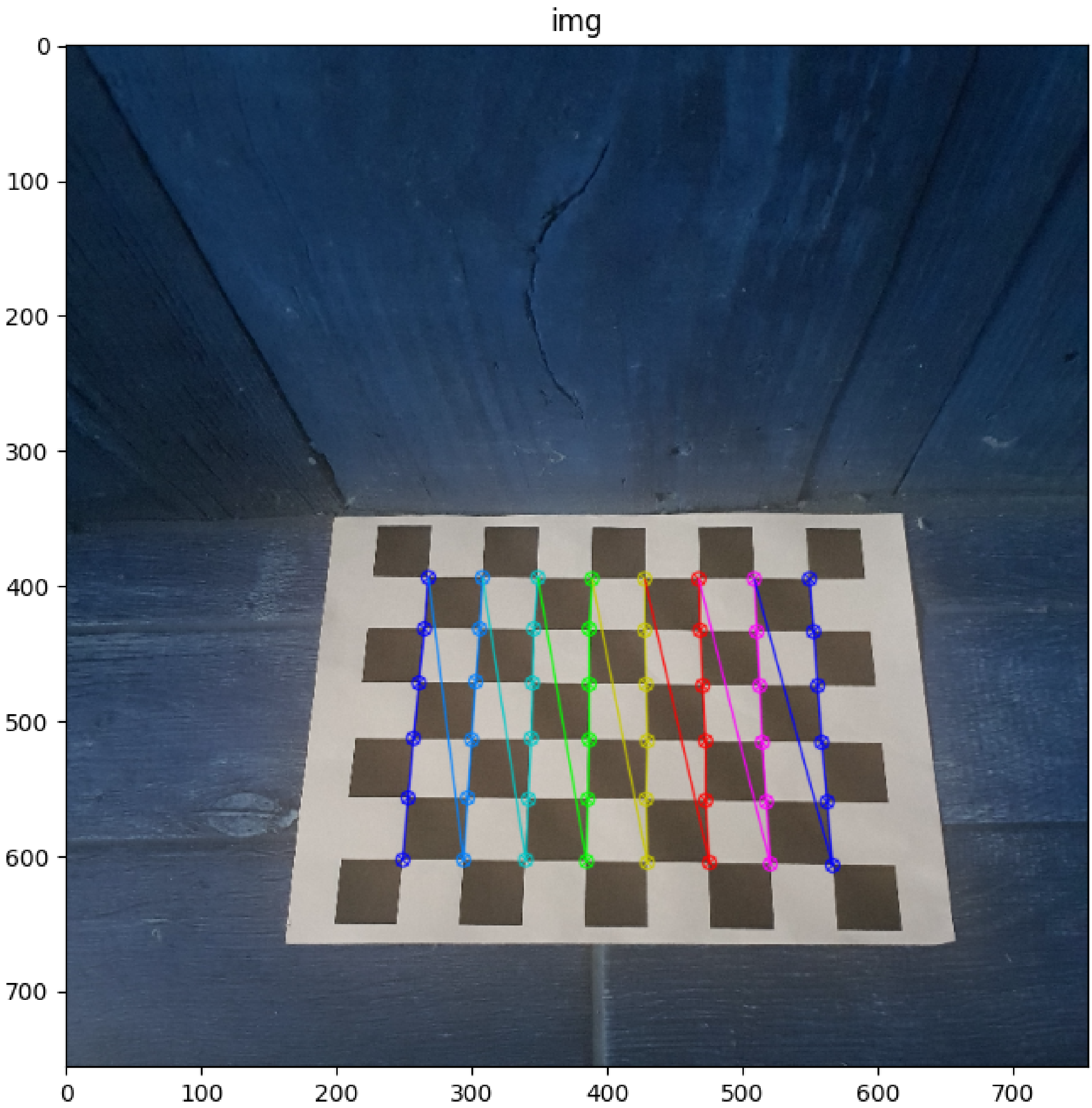
```
plt.imshow(octave5_d)
plt.title('error')
plt.show()
```

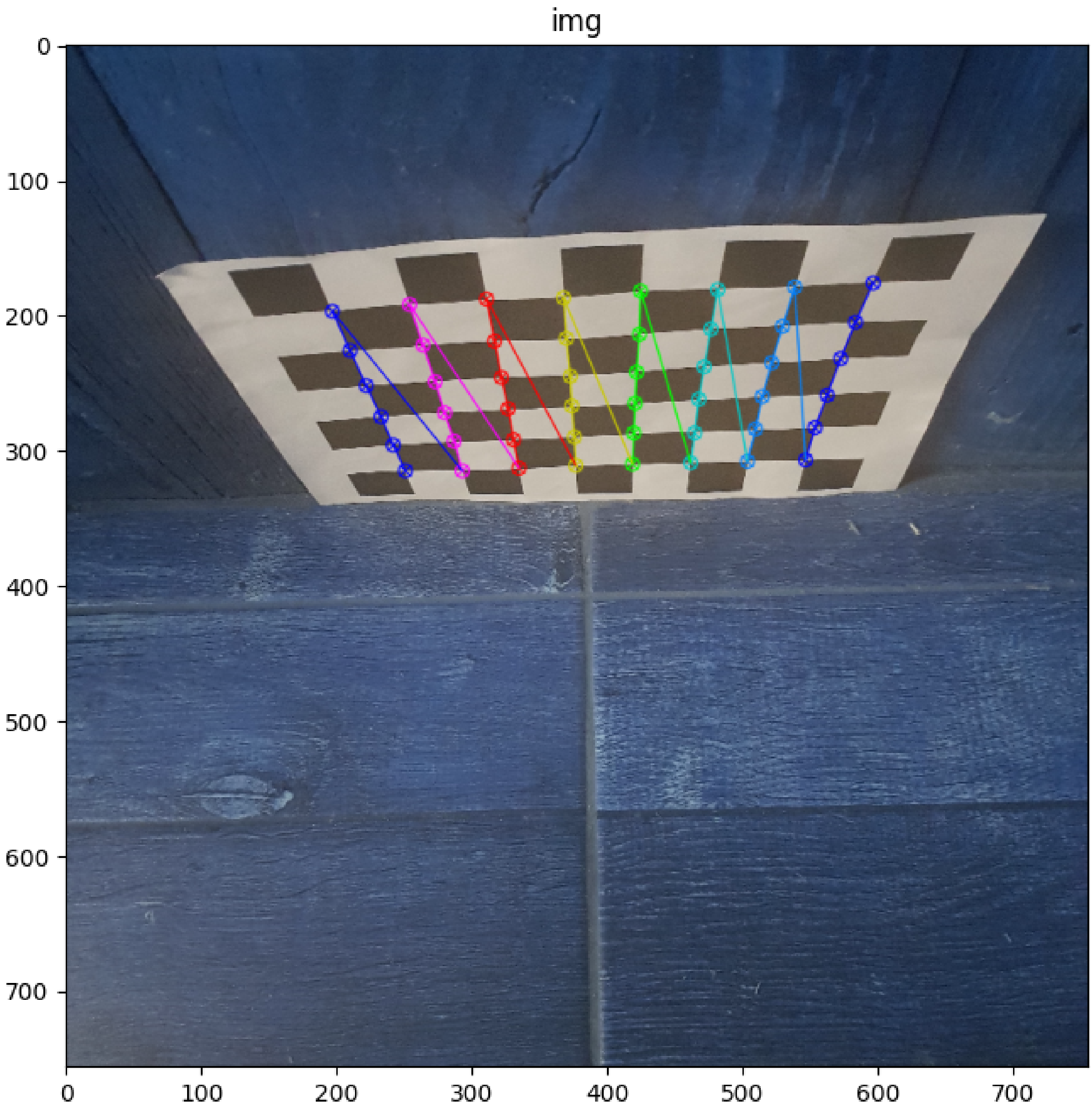


so we obtain fx = 3909 and fy = 3884 using Octave

Python

In [41]:
```
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)

objpoints = []
imgpoints = []
images = ['image5.jpg', 'image6.jpg']

image_corners = []


for fname in images:
    img = cv2.imread(fname)
    scale_percent = 25 # percent of original size
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)

    # resize image
    img = cv2.resize(img, dim, interpolation = cv2.INTER_NEAREST)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (6,8), None)
    if ret == True:
        image_corners.append(corners)
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        cv2.drawChessboardCorners(img, (6,8), corners2, ret)
        plt.imshow(img)
        plt.title('img')
        plt.show()
```


img
0
100
200
300
400
500
600
700
0
100
200
300
400
500
600
700



In [42]:

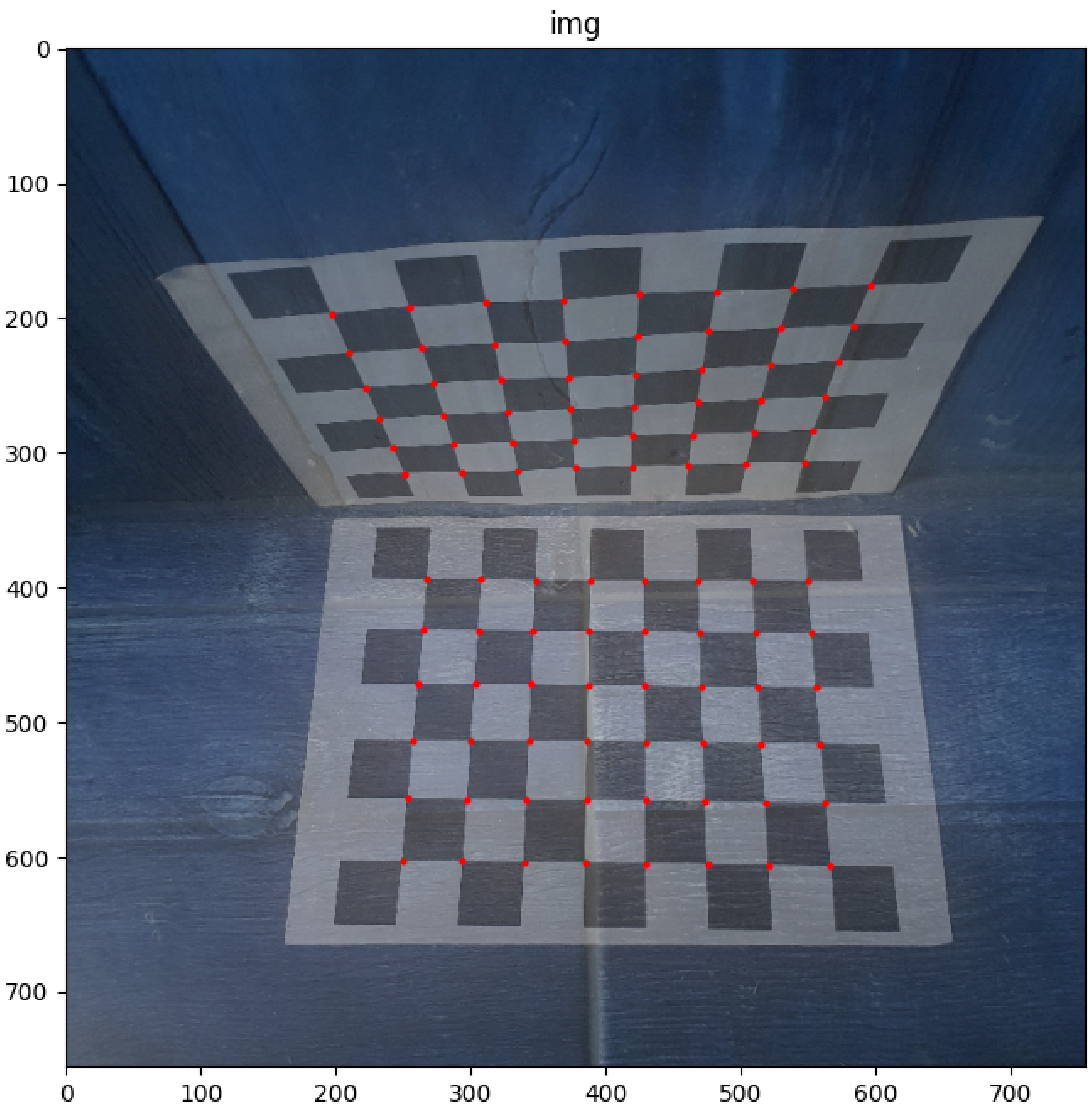
```
x = []
y = []

for i in range(len(image_corners[0])):
    x = np.append(x, image_corners[0][i][0][0])
    y = np.append(y, image_corners[0][i][0][1])
for i in range(len(image_corners[1])):
    x = np.append(x, image_corners[1][i][0][0])
    y = np.append(y, image_corners[1][i][0][1])

points = [x, y]
```

The images are rotated about 90° of each other. We can combine the two images and the points of the chess board to obtain the points that will be used by the algorithm.

In [43]:

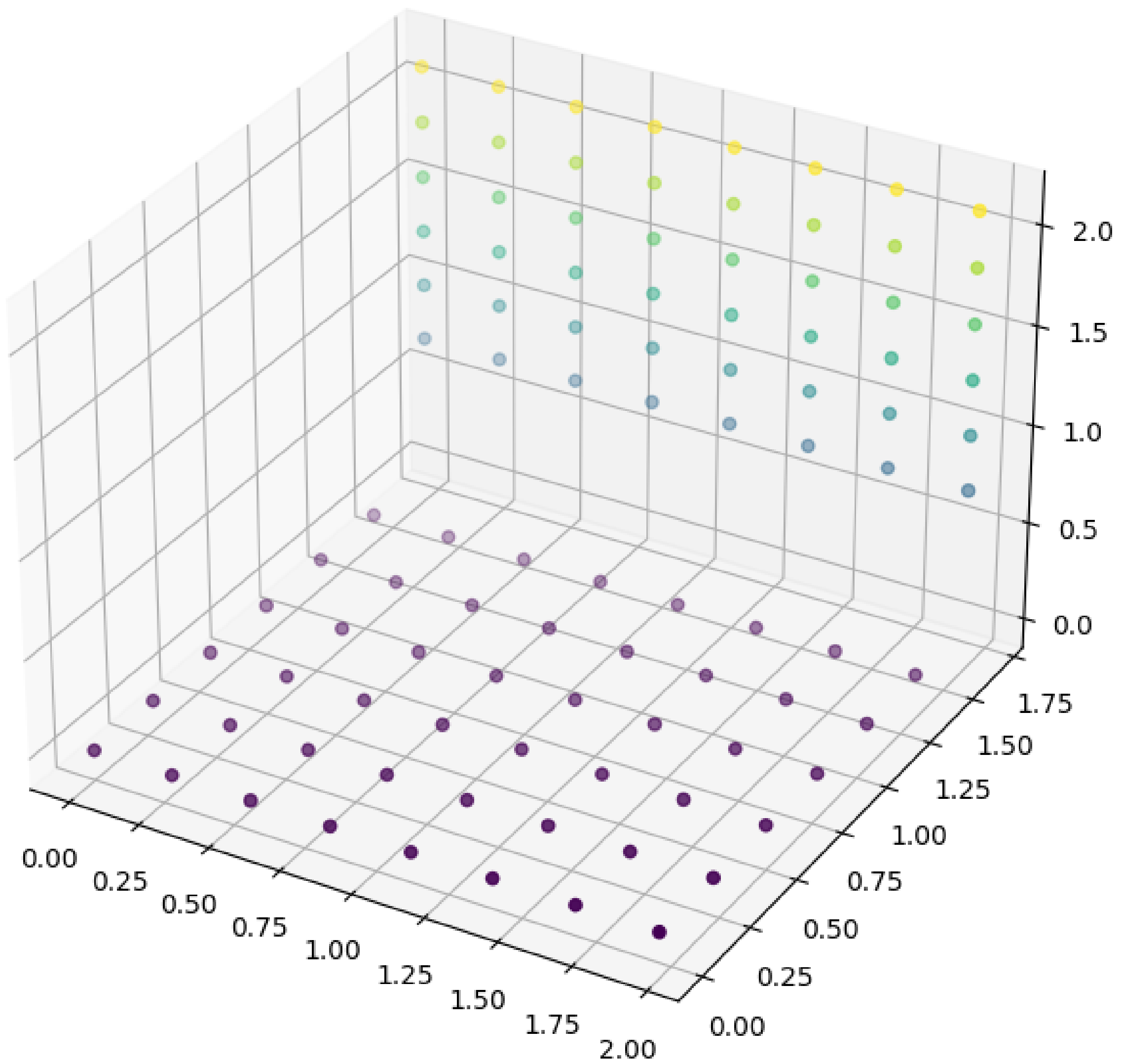
```
img6 = cv2.imread('image6.jpg')
scale_percent = 25
width = int(img6.shape[1] * scale_percent / 100)
height = int(img6.shape[0] * scale_percent / 100)
dim = (width, height)
img6 = cv2.resize(img6, dim, interpolation = cv2.INTER_NEAREST)

img5 = cv2.imread('image5.jpg')
```

```
scale_percent = 25
width = int(img5.shape[1] * scale_percent / 100)
height = int(img5.shape[0] * scale_percent / 100)
dim = (width, height)
img5 = cv2.resize(img5, dim, interpolation = cv2.INTER_NEAREST)

def blend(list_images):
    equal_fraction = 1.0 / (len(list_images))
    output = np.zeros_like(list_images[0])
    for img in list_images:
        output = output + img * equal_fraction
    output = output.astype(np.uint8)
    return output

list_images = [img6, img5]
img = blend(list_images)

plt.imshow(img)
plt.title('img')
plt.scatter(x=points[0], y=points[1], c='r', s=4)
plt.show()
```



The points are obtained by setting the chess board of the first image as the (1,1,0) plane, and the chess board of the second image as the (1,0,1) plane.

In [44]:
```
p = image_corners[0]

square_x_cm = 0.28
square_y_cm = 0.28

image_point = []
world_point = []

# Get the points of the first image in world coordinates by setting the z coo
# and varying the coordinates in the x,y direction.
for i in range(8):
    for j in range(6):
        image_point.append(
            p[i*6 + j][0]
        )
        world_point.append([
            i*square_x_cm, # x
            j*square_y_cm, # y
            0.0            # z
        ])


p = image_corners[1]

# Get the points of the second image in world coordinates by setting the y co
# because the points in the second plane are shiffted approximately 6 squares
# and varying the coordinates in the x,y direction.
for i in range(8):
    for j in range(6):
        image_point.append(
            p[i*6 + j][0]
        )
        world_point.append([
            7*square_x_cm - i*square_x_cm,                          # x
            6*square_y_cm,                                          # y
            0.15 + square_y_cm + 6*square_y_cm - j*square_y_cm # z
        ])

xdata = []
ydata = []
zdata = []

for i in range(96):
    xdata = np.append(xdata, world_point[i][0])
    ydata = np.append(ydata, world_point[i][1])
    zdata = np.append(zdata, world_point[i][2])

ax = plt.axes(projection='3d')

ax.scatter3D(xdata, ydata, zdata, c=zdata);
```



In [45]:

```
A = []

def build_line(image_points, world_points, i):
    return [
        image_points[i][0]*world_points[i][0],
        image_points[i][0]*world_points[i][1],
        image_points[i][0]*world_points[i][2],
        image_points[i][0],
        -1*image_points[i][1]*world_points[i][0],
        -1*image_points[i][1]*world_points[i][1],
        -1*image_points[i][1]*world_points[i][2],
        -1*image_points[i][1],
    ]

for i in range(96):
    A.append(build_line(image_point, world_point, i))

A = np.reshape(A,(96,8))

SVD = np.linalg.svd(A, full_matrices=True)
SVD[1]
```

Out[45]:

```
array([1.10676684e+04, 4.45145196e+03, 2.85267082e+03, 1.71392022e+03,
       8.30897869e+02, 3.61115696e+02, 2.97429441e+02, 7.98337546e+00])
```

The smallest eigen value is in the 7'th row, so the vector that we are interested in is:

In [46]:
```
v = SVD[2][SVD[1].argmin()]

print("min eigen value =", min(SVD[1]))
print("corresponding eigen vector =", v)
```

```
min eigen value = 7.983375462342892
corresponding eigen vector = [-0.00698853 -0.16202727 -0.18580742  0.86300794
0.2220594   0.05278596
 -0.11649634  0.35878   ]
```

The gamma parameter, given by Trucco and Verri, is given by:

In [47]:
```
y = np.sqrt(v[0]**2 + v[1]**2 + v[2]**2)
y
```

Out[47]:
```
0.24662942157525194
```

The alpha*gamma parameter, given by Trucco and Verri, is given by:

In [48]:
```
ay = np.sqrt(v[4]**2 + v[5]**2 + v[6]**2)
ay
```

Out[48]:
```
0.25625794407960917
```

And so we have the alpha(aspect ratio) parameter given by:

In [49]:
```
a = ay/y
a
```

Out[49]:
```
1.0390404455513
```

The first two rows of the intrinsic matrix can by recovered by:

In [50]:
```
r = np.zeros((3,3))

r[1,0] = v[0]
r[1,1] = v[1]
r[1,2] = v[2]
Ty     = v[3]
r[0,0] = v[4]/a
r[0,1] = v[5]/a
r[0,2] = v[6]/a
Tx     = v[7]/a
r
```

Out[50]:
```
array([[ 0.21371584,  0.0508026 , -0.11211916],
       [-0.00698853, -0.16202727, -0.18580742],
       [ 0.        ,  0.        ,  0.        ]])
```

The Tz and fx parameters of the third row can be approximated by:

In [51]:
```
def build_A_line(image_points, world_points, r, i):
    return [
        image_points[i][0], (r[0,0]*world_points[i][0] + r[0,1]*world_points[
    ]

A = []
```

```
b = []

for i in range(96):
    A.append(build_A_line(image_point, world_point, r, i))

A = np.reshape(A,(96,2))

def build_b_line(image_points, world_points, r, i):
    return [
        -1*image_points[i][0]*(r[2,0]*world_points[i][0] + r[2,1]*world_point
    ]

for i in range(96):
    b.append(build_b_line(image_point, world_point, r, i))

((np.linalg.inv(A.T.dot(A))).dot(A.T)).dot(b)
```

Out[51]:

```
array([[0.],
       [0.]])
```