Reflection Essay In lab05

First we clocked 2 pictures of the dot pattern in class

Image 1 of pattern

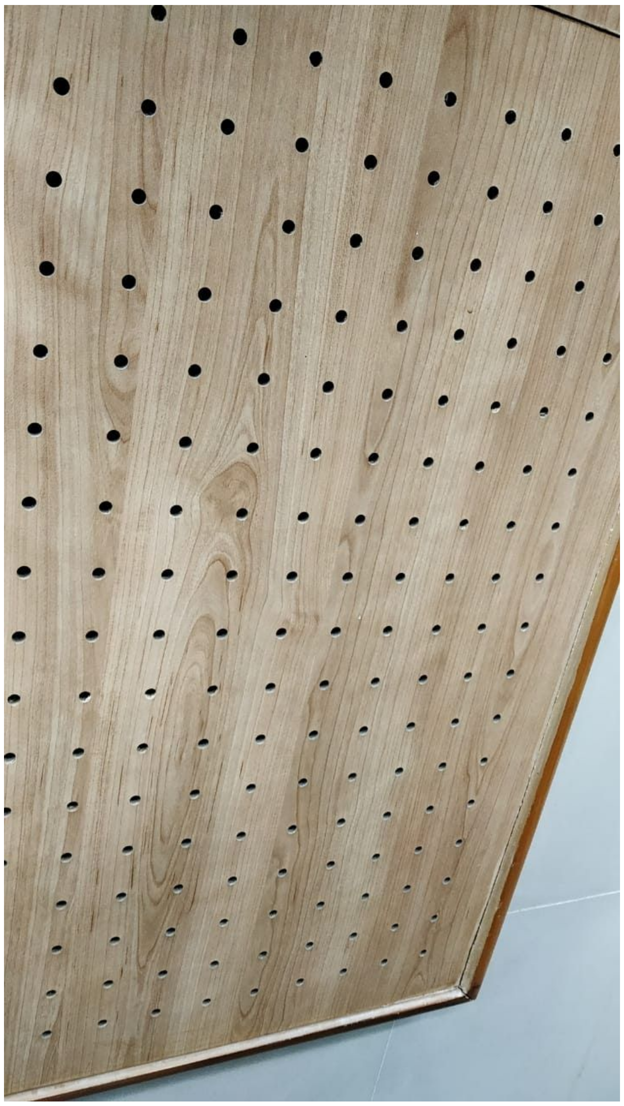
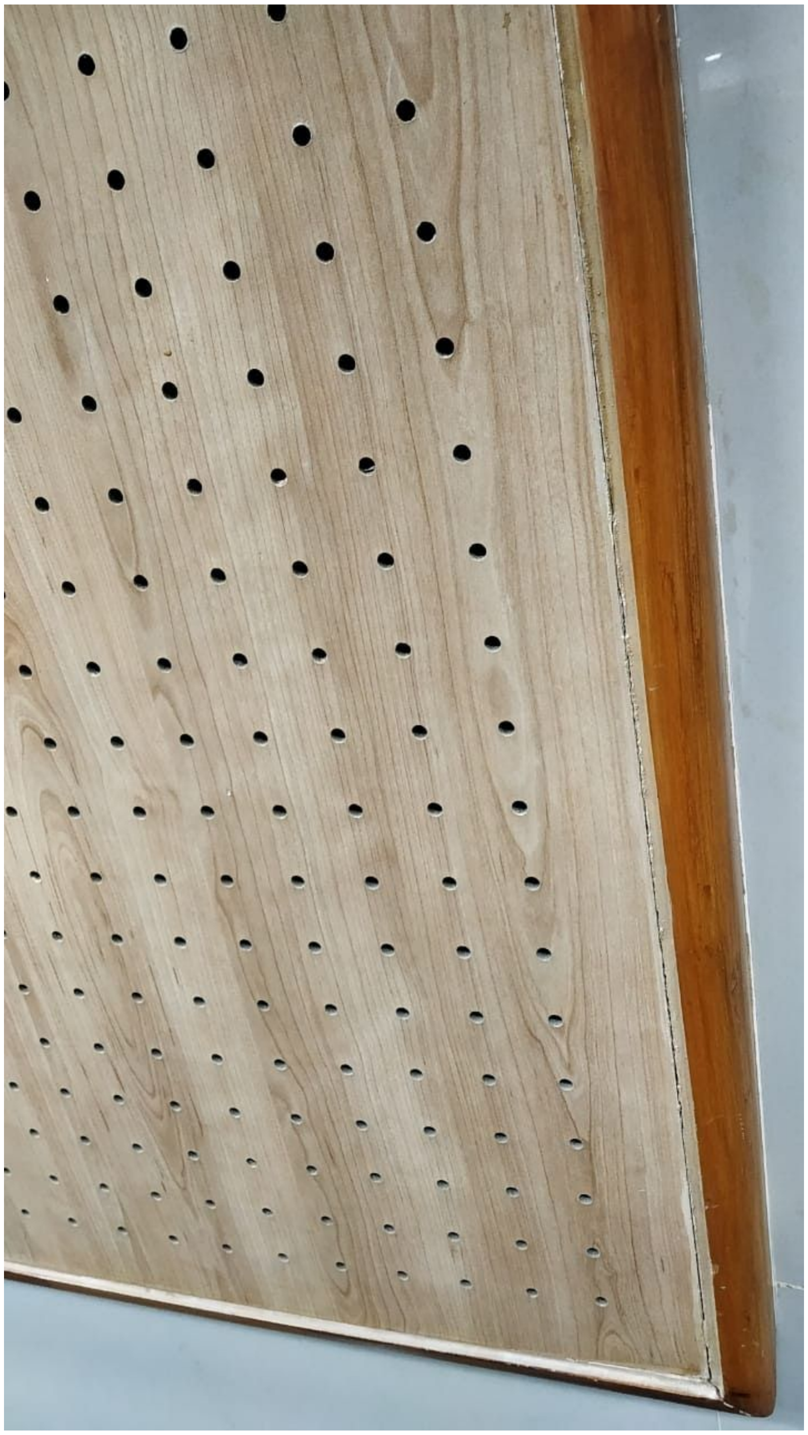
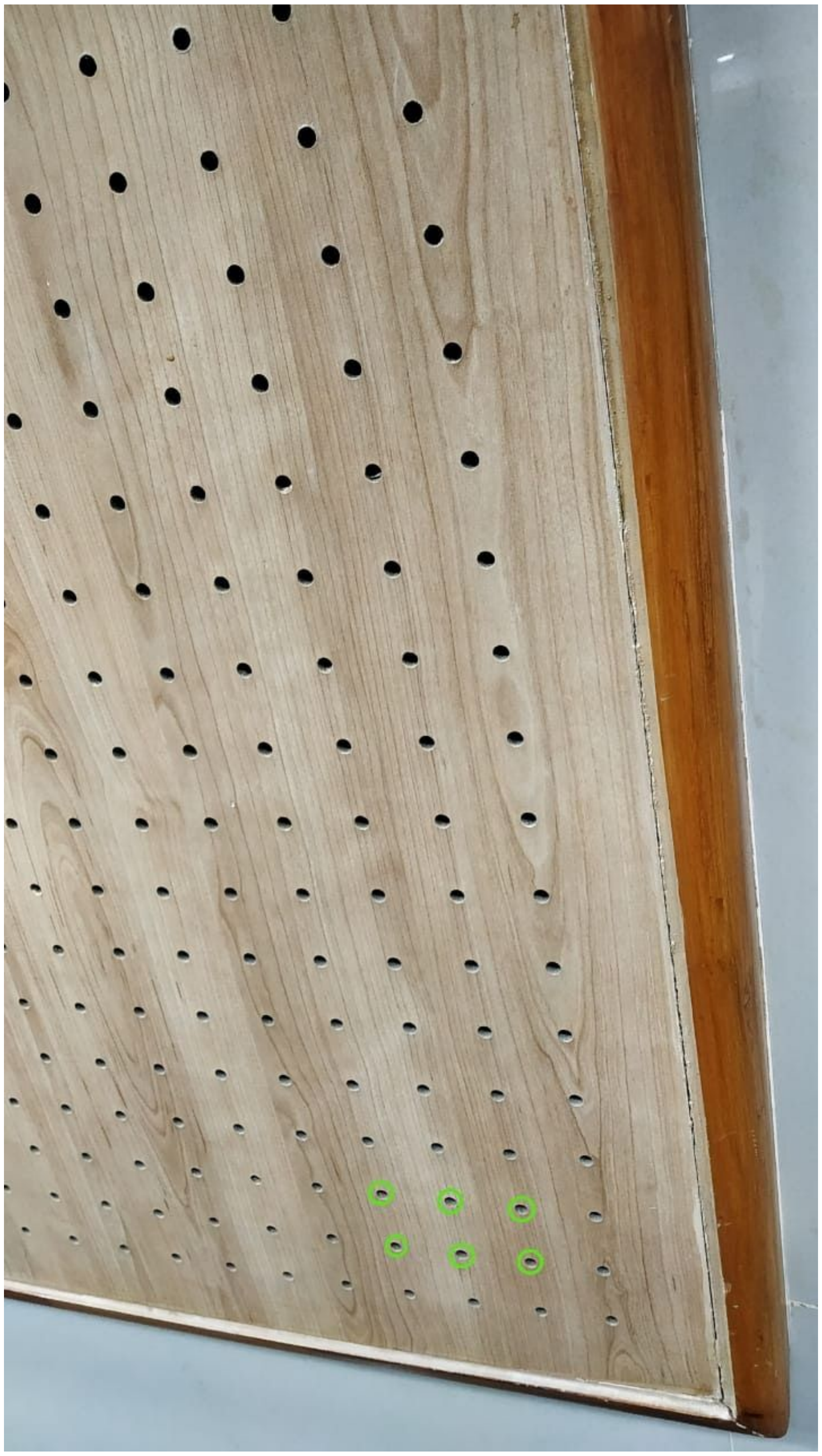


Image 2 of pattern



Then we highlighted the chosen points
true.jpeg



Next we determined the pointwise correspondences and got the below mapping of points

```
objpoints = np.array([[[0, 0, 0], [3, 0, 0], [6, 0, 0], [0, 3, 0], [3, 3, 0], [6, 3, 0]],
                    [[0, 0, 0], [3, 0, 0], [6, 0, 0], [0, 3, 0], [3, 3, 0], [6, 3, 0]]],dtype = np.float32)

imgpoints = np.array([[[466,1111],[406,1104],[345,1097],[456,1063],[398,1058],[335,1051]],
                    [[539,1162],[477,1155],[417,1147],[533,1119],[466,1111],[408,1105]]],dtype =
np.float32)
```

Next we performed Camera Calibration using these 12 points from 2 images and got the below results (only parameters discussed in class are shown)

Camera Intrinsic matrix (Focal Length and scaling)

```
[[8.95102415e+03 0.00000000e+00 3.32993104e+02]
 [0.00000000e+00 4.18822619e+03 6.16443847e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Rotation Vectors (Rotational mapping, different for both the pictures)

Image 1

```
[array([[-1.51560673],
        [ 0.02767937],
        [-2.76030216]]),
```

Image 2

```
array([[-1.49936923],
        [ 0.04287658],
        [-2.7573874 ]])]
```

Translation Vectors (Different for both the pictures)

Camera Translation vector

```
[array([[  2.47827869],
        [ 28.43711669],
        [233.95737768]]),

array([[  4.2300328 ],
        [ 31.93079002],
        [234.70441726]])]
```

Successful?

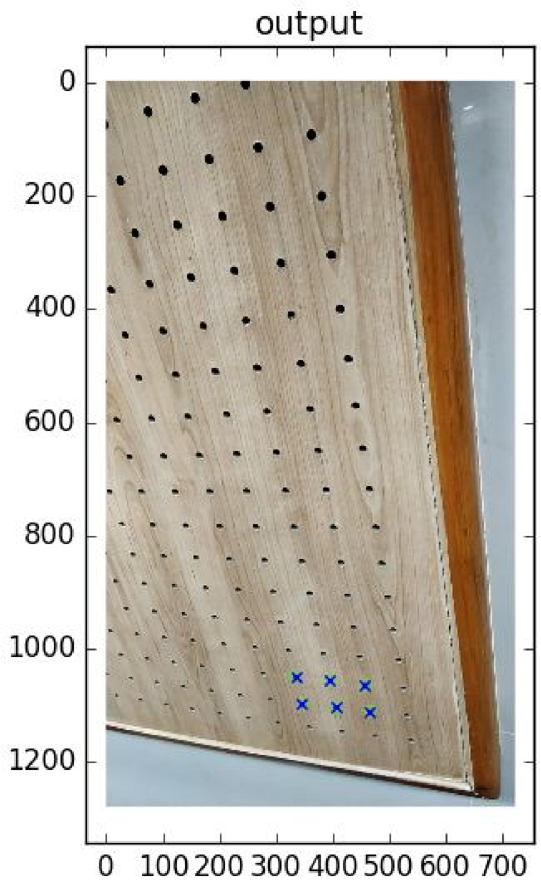
We used cv2.projectPoints() and got the reconstructed points as below

```
Image 1
Projected
[[[ 466.22314 1110.6914 ]
  [ 405.04352 1103.9945 ]
  [ 345.7455  1097.2542 ]
  [ 457.38522 1063.6304 ]
  [ 395.68304 1057.3499 ]
  [ 335.93024 1051.075  ]]]
True
np.array([[[466,1111],[406,1104],[345,1097],[456,1063],[398,1058],[335,1051]],

Projected
[[[ 539.3025  1161.9518 ]
  [ 477.09747 1154.7448 ]
  [ 416.58838 1147.3684 ]
  [ 531.4719  1118.8105 ]
  [ 468.39566 1111.6713 ]
  [ 407.14108 1104.459  ]]]

True
 [[539,1162],[477,1155],[417,1147],[533,1119],[466,1111],[408,1105]]],dtype = np.float32)
```

average projection error 1.07

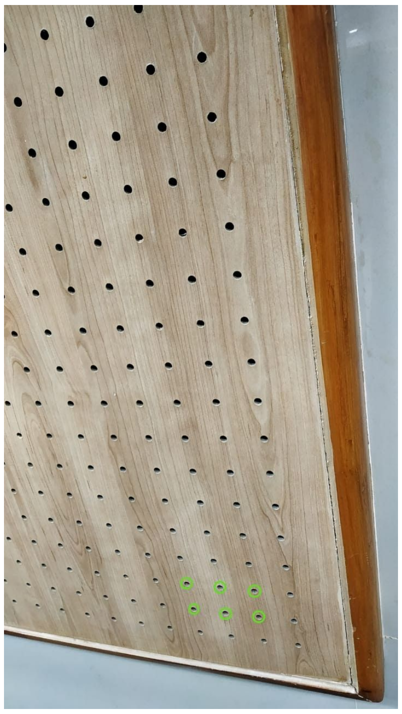


The matchings are quite close as can be seen from the lower right corner of the image

Lab 05: Camera Outlab

In the outlab problem, we did not have to read in any images, instead we were given some image coordinates and we needed to find the camera matrix.
Next similar to the inlab we were to use the camera matrix to reproject the real world coordinates onto the sensor and obtain the **reconstructed** image coordinates.

The primary challenge in this problem was that the points were scrambled in a random order. First we had to programmatically reorder them into the order we used in the **inlab**



In the inlab we followed the following order

1. Start from lower right corner and move left
2. Once at the end of lower row, go to right most element of upper row

The real world coordinates of the points when we follow this order is:
objpoints = np.array([[[0, 0, 0], [3, 0, 0], [6, 0, 0], [0, 3, 0], [3, 3, 0], [6, 3, 0]],
[[0, 0, 0], [3, 0, 0], [6, 0, 0], [0, 3, 0], [3, 3, 0], [6, 3, 0]]]

The raw points given in points.txt were
Image 1
443 359
390 360
392 417
498 356
444 417
498 415
Image 2
575 396
513 396
449 331
513 330
450 397
576 329

We re-arranged the points through code to get

Image 1
[[498, 356], [443, 359], [390, 360], [498, 415], [444, 417], [392, 417]]

Image 2
[[576, 329], [513, 330], [449, 331], [575, 396], [513, 396], [450, 397]]

Next Challenge: Unknown Image Size

The true image-size was unknown in the given program.

If we were to compute the camera matrix using a single set of 6 points from just 1 image, then there is almost no optimization/search/learning of intrinsic camera parameters involved.

But in general, when we pass many points from multiple images, the caliberateCamera( ) method of openCV performs an optimization procedure to solve for the camera matrix.
In particular it looks to **minimize the norm** of a certain vector (since the system is overdetermined)

This is consistent with the re-projection error being 0.0 irrespective of the image dimensions used in the case when only 6 points from 1 image are passed.

For the optimization procedure in case of passing multiple points (2 in our case), the calibration algorithm uses the image size as an initialization for the coordinates of the principal point offset (x_o, y_o) and the initial value ends up heavily biasing the final camera matrix.

Based on different initializations, the accuracy of the final result will vary slightly and we will get slightly different error values.

For example we tried the following image sizes (The principal offsets are computed as size_x/2, size_y/2)

(width, height) - Projection Error
(1000, 1000) - 0.04
(800, 600) - 0.08
(1280, 720) - 0.14
(720,1280) - 0.14
# Some **nonsensical** sizes (since sensor coordinates are larger than the size which is not correct)
(200, 200) - 0.20
(1, 1) - 0.21

The variation here is still quite significant across sizes
A possible reason for this can be the **camera distortion parameters** fitted by the API.
The ideal camera model does not have distortion, but the API fits an 8 coefficient distortion model for **tangential and radial** distortion.
The effect of these parameters during reprojection onto camera sensor depend on the image size (through image center) significantly.
The below are the error values for the same cases with the **effect of these turned off** using API flags.

(width, height) - Projection Error
(1000, 1000) - 0.17
(800, 600) - 0.17
(1280, 720) - 0.17
(720,1280) - 0.17
# Some **nonsensical** sizes (since sensor coordinates are larger than the size which is not correct)
(200, 200) - 0.23
(1, 1) - 0.23

We see that here the effect of image size has almost vanished except for the **nonsensical** cases where the behaviour of the **API can be assumed to be unreliable**