All the code and solutions available at https://github.com/rezmansouri/csc8830. Web applications are executable at https://rezmansouri.github.io/csc8830.

1. Report the calibration matrix for the camera chosen and verify (using an example) the same.

The calibration matrix, according to MATLAB's camera calibration toolbox result for the oak-d LITE camera is as follows (Please refer to appendix I for the calibration footage):

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

```
Focal length (pixels):    [ 1434.4141 +/- 6.8278     1430.6807 +/- 6.7548  ]
Principal point (pixels):[  949.7754 +/- 1.6231      541.4124 +/- 0.8631  ]
```

According to Eq. 1 and the results, the calibration matrix would be estimated as:

$$K = \begin{bmatrix} 1434.41 & 0 & 949.77 \\ 0 & 1430.68 & 541.41 \\ 0 & 0 & 1 \end{bmatrix}$$

To verify this calibration matrix, I perform perspective projection. I will take a picture of an object, consider some points of it, and compare resulting pixel coordinates of pre-multiplying the calibration matrix (K) with the points' coordinates with respect to camera (X_c), with the respective pixel coordinates in the image. In short, I will be testing the results of the following equation for multiple points, by comparing them to true pixel coordinates in the image.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = K \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (2)$$

First, in order to obtain X_c, I will define the camera coordinate system (the axes). Here's the experimentation setup:

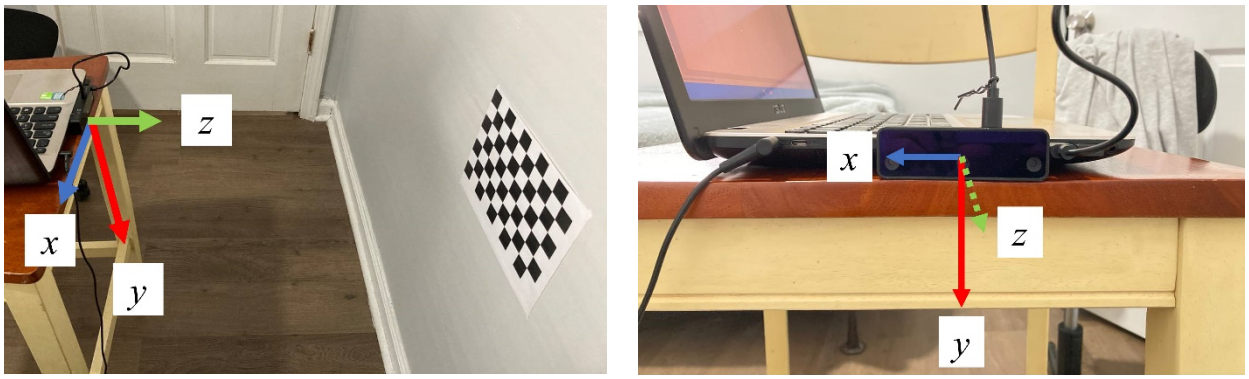


Figure 1. Experiment's setup. Camera's coordinate system (the axes)

Second, I will take an image of an object which I can easily measure coordinates of my points of choice. A checkerboard with square side length of 26mm is my choice. I will place the checkerboard in front of the camera such that the camera coordinate frame's principal point is aligned with the top left corner of its second square from the top and left. Figure 2 shows how the checkerboard is aligned with the camera coordinate system's axes. The camera is placed 68.5cm (685mm) from the object, thus, every point on the checkerboard will have $z_c = 685$.

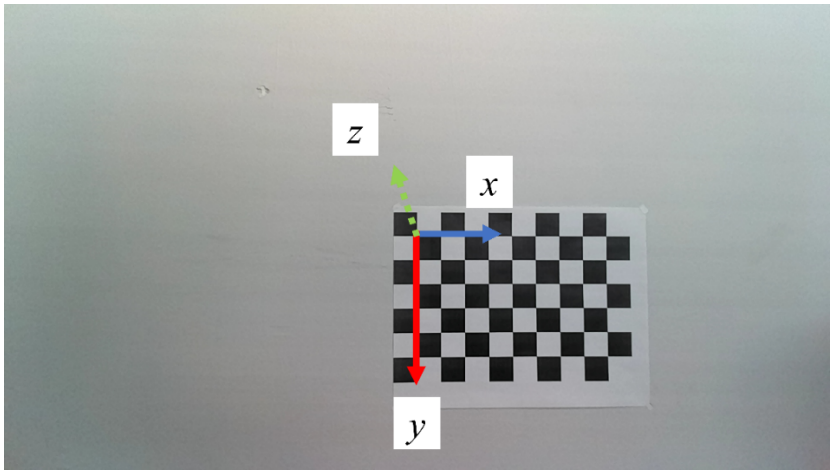


Figure 2. Alignment of the reference object to camera coordinate system axes

EXAMPLE 1: A point at (52, 52, 685) at camera coordinate frame

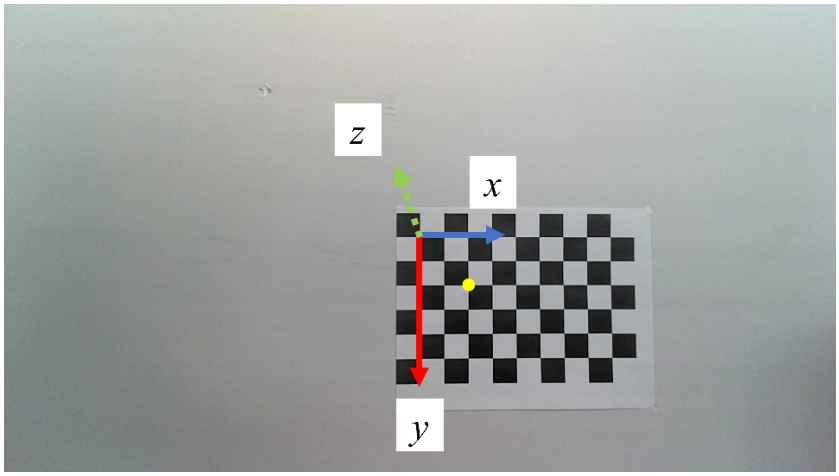


Figure 3. A reference point at (52, 52, 685) – The yellow dot

$$\begin{bmatrix} 1434.41 & 0 & 949.77 \\ 0 & 1430.68 & 541.41 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 52 \\ 52 \\ 685 \end{bmatrix} = \begin{bmatrix} 1434.41 * 52 + 0 * 52 + 949.77 * 685 \\ 0 * 52 + 1430.68 * 52 + 541.41 * 685 \\ 0 * 52 + 0 * 52 + 1 * 685 \end{bmatrix} = \begin{bmatrix} 725181.77 \\ 445261.21 \\ 685 \end{bmatrix} \equiv \begin{bmatrix} 1058.66 \\ 650.01 \\ 1 \end{bmatrix}$$

This point has true pixel coordinates of (1071, 648). Comparing it with the perspective projection equation's result, and neglecting small differences due to measurement errors, the calibration matrix is verified. $(1058.66, 650.01) \cong (1071, 648)$. Figure 4 shows both the calculated point coordinates and its true pixel coordinates.

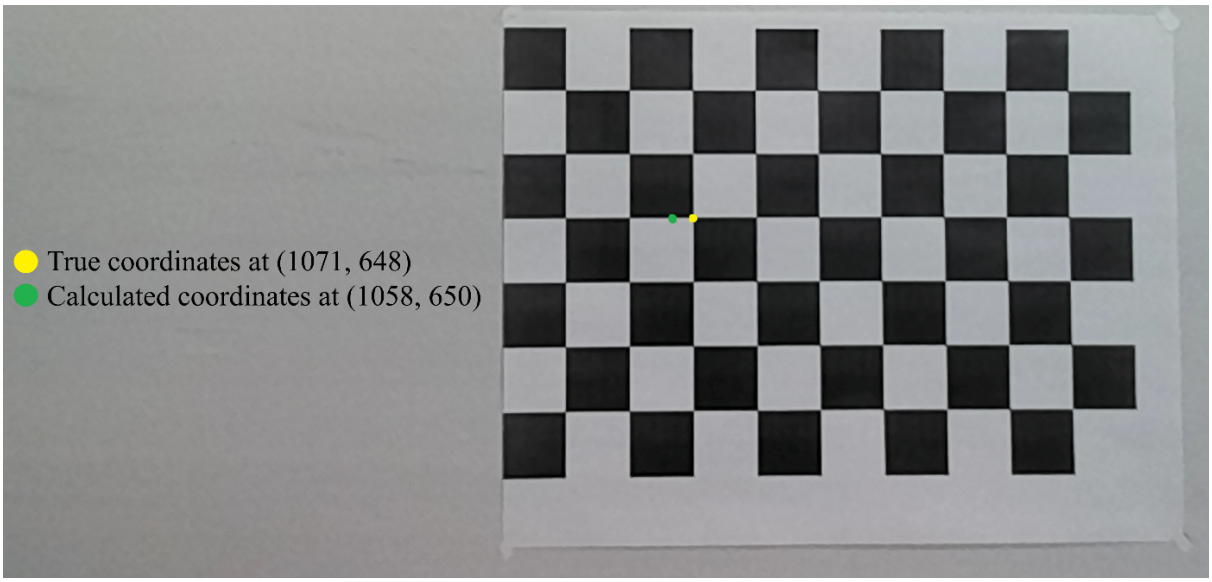


Figure 4. Comparison between true coordinates and calculated coordinates in example 1

EXAMPLE 2: A point at (-26, 78, 685) at camera coordinate frame

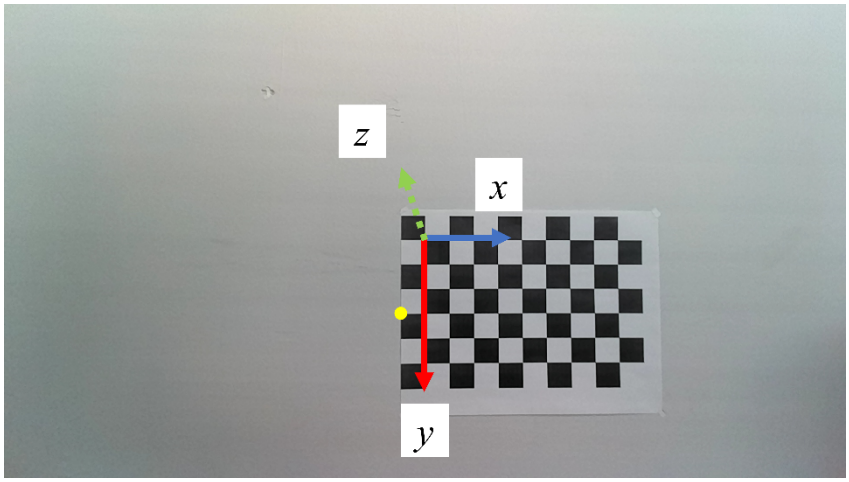


Figure 5. A reference point at (-26, 78, 685) – The yellow dot

$$\begin{bmatrix} 1434.41 & 0 & 949.77 \\ 0 & 1430.68 & 541.41 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -26 \\ 78 \\ 685 \end{bmatrix} = \begin{bmatrix} 1434.41 * (-26) + 0 * 78 + 949.77 * 685 \\ 0 * (-26) + 1430.68 * 78 + 541.41 * 685 \\ 0 * (-26) + 0 * 78 + 1 * 685 \end{bmatrix} = \begin{bmatrix} 613297.79 \\ 482458.89 \\ 685 \end{bmatrix} \equiv \begin{bmatrix} 895.32 \\ 704.31 \\ 1 \end{bmatrix}$$

The true pixel coordinates of this reference points are (904, 705). Similar to example 1, the calibration matrix is verified. $(895.32, 704.31) \cong (904, 705)$. Figure 6 shows both the calculated point coordinates and its true pixel coordinates.

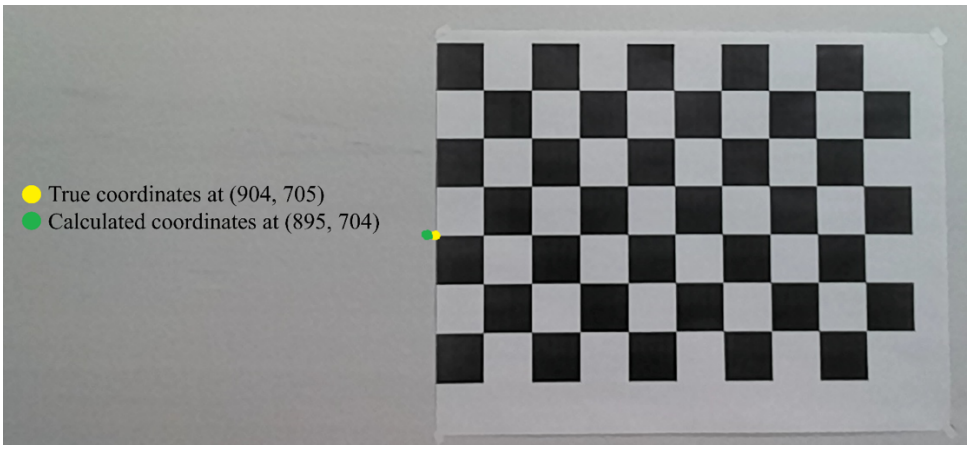


Figure 6. Comparison between true coordinates and calculated coordinates in example 2

---

2. Point the camera to a chessboard pattern or any known set of reference points that lie on the same plane. Capture a series of 10 images by changing the orientation of the camera in each iteration. Select any 1 image, and using the image formation pipeline equation, set up the linear equations in matrix form and solve for intrinsic and extrinsic parameters (extrinsic for that particular orientation). You will need to make measurements of the actual 3D world points, and mark pixel coordinates. Once you compute the Rotation matrix, you also need to compute the angles of rotation along each axis. Choose your order of rotation based on your experimentation setup.

The approach that I will take is as follows. The equation below represents the knowns and unknowns:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

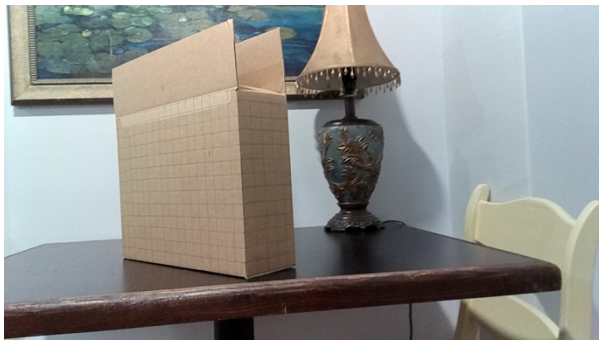
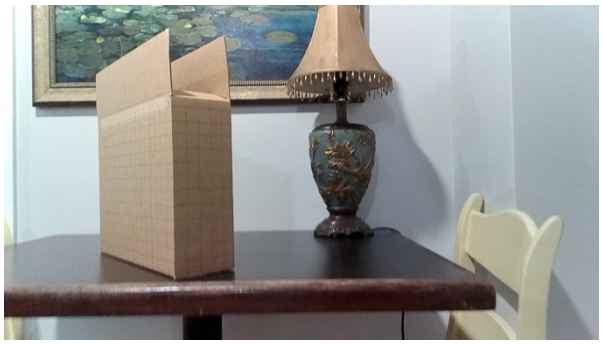
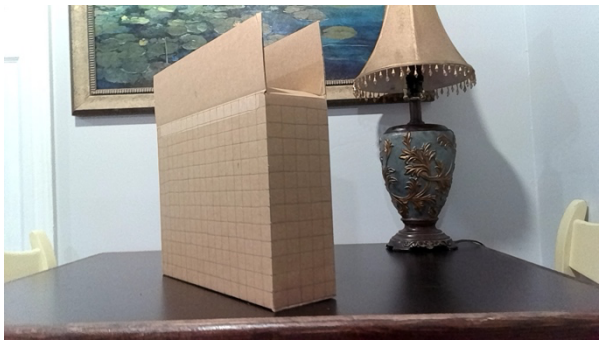
With u, v being the pixel coordinates (known), x, y and z world coordinates of the reference points (known) and P as the projection matrix (unknown). First the effort is to solve for P. p_{11} to p_{33} are unknown and p_{34} is a scaling factor. With eleven unknowns, 6 reference points are required as each reference point gives us two equations. Here are the equations in matrix form.

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_6 & y_6 & z_6 & 1 & 0 & 0 & 0 & 0 & -u_6x_6 & -u_6y_6 & -u_6z_6 & -u_6 \\ 0 & 0 & 0 & 0 & x_6 & y_6 & z_6 & 1 & -v_6x_6 & -v_6y_6 & -v_6z_6 & -v_6 \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

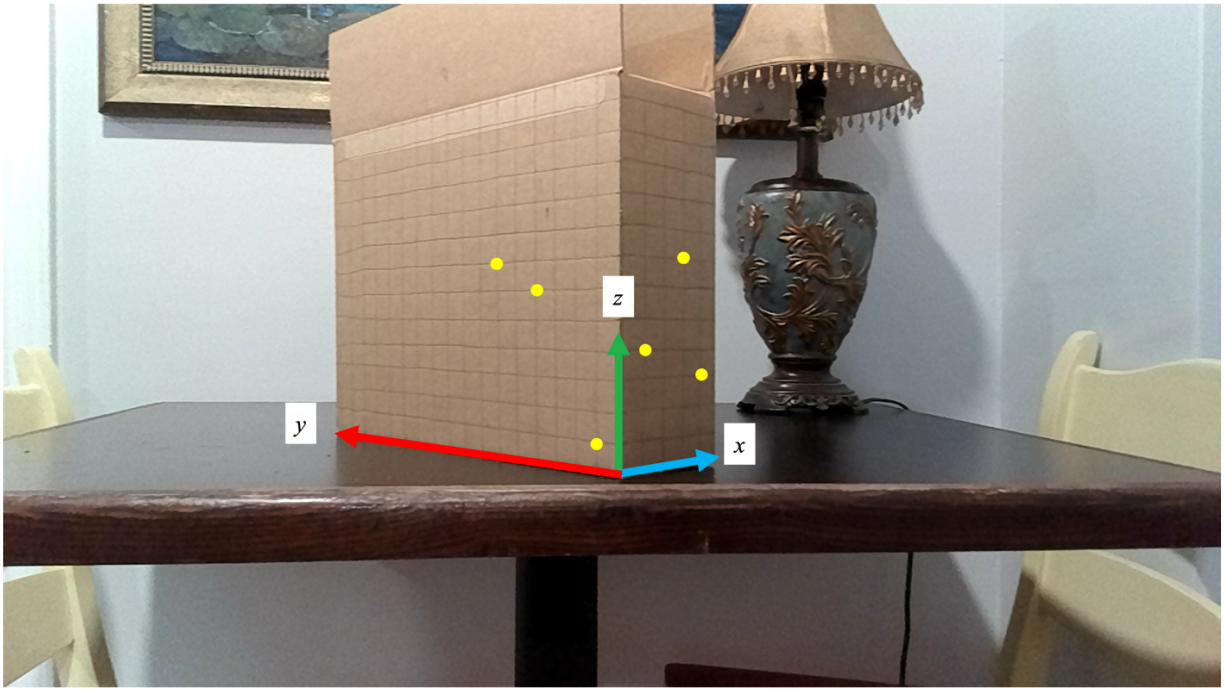
Calling the equations' matrix A, in order to solve for P, the problem is equivalent to the constrained least square problem of A^TA. Thus, P will be the corresponding eigenvector of the minimum eigenvalue of A^TA. Next, I will decompose $\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$ into K (Calibration matrix) and the Rotation matrix (R) using QR factorization, and I will compute t (translation vector) by $K^{-1}\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$.

I tried to solve this for a set of reference points that lie on the same plane, but the points would all have the same coordinate along one of the axes (they are coplanar) leading to a singular matrix A, leading to incorrect values further. Thus, as presented in Dr. Nayyar's lecture for camera calibration, I will choose my reference points from a 3D object with chessboard pattern. Here are the 10 Images:




And this is the image I will proceed with, with the reference points and the world coordinate system axes depicted:



Each square length on the card-box has side length of 20cm. The world coordinates of the reference points are:

$$\begin{bmatrix}20\\0\\80\end{bmatrix},\begin{bmatrix}0\\120\\140\end{bmatrix},\begin{bmatrix}0\\20\\20\end{bmatrix},\begin{bmatrix}60\\0\\140\end{bmatrix},\begin{bmatrix}80\\0\\60\end{bmatrix},\begin{bmatrix}0\\80\\120\end{bmatrix}$$

And the image pixel coordinates:

$$\begin{bmatrix}1011\\552\end{bmatrix},\begin{bmatrix}778\\414\end{bmatrix},\begin{bmatrix}938\\696\end{bmatrix},\begin{bmatrix}1075\\409\end{bmatrix},\begin{bmatrix}1100\\591\end{bmatrix},\begin{bmatrix}839\\455\end{bmatrix}$$

The system of equations will be:

$$\begin{bmatrix}20&0&80&1&0&0&0&0&-20220&0&-80880&-1011\\0&0&0&0&20&0&80&1&-11040&0&-44160&-552\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&80&120&1&0&0&0&0&0&-67120&-100680&-839\\0&0&0&0&0&80&120&1&0&-36400&-54600&-455\end{bmatrix}\begin{bmatrix}p_{11}\\p_{12}\\p_{13}\\p_{14}\\p_{21}\\p_{22}\\p_{23}\\p_{24}\\p_{31}\\p_{32}\\p_{33}\\p_{34}\end{bmatrix}=\begin{bmatrix}0\\\vdots\\0\end{bmatrix}$$

Solving for P via constrained least squares for A^TA:

$$P=\begin{bmatrix}-2.34\times10^{-3}&5.07\times10^{-4}&-1.69\times10^{-6}&-7.92\times10^{-1}\\-4.68\times10^{-4}&-5.15\times10^{-4}&2.03\times10^{-3}&-6.10\times10^{-1}\\-1.00\times10^{-6}&-1.10\times10^{-6}&3.34\times10^{-8}&-8.12\times10^{-4}\end{bmatrix}$$

Verifying P for the first reference point:

$$\begin{bmatrix}-2.34\times10^{-3}&5.07\times10^{-4}&-1.69\times10^{-6}&-7.92\times10^{-1}\\-4.68\times10^{-4}&-5.15\times10^{-4}&2.03\times10^{-3}&-6.10\times10^{-1}\\-1.00\times10^{-6}&-1.10\times10^{-6}&3.34\times10^{-8}&-8.12\times10^{-4}\end{bmatrix}\begin{bmatrix}20\\0\\80\\1\end{bmatrix}\begin{bmatrix}-8.39\times10^{-1}\\-4.58\times10^{-1}\\-8.30\times10^{-4}\end{bmatrix}\equiv\begin{bmatrix}1.01\times10^{3}\\5.51\times10^{2}\\1\end{bmatrix}\approx\begin{bmatrix}1011\\552\\1\end{bmatrix}$$

Decomposing the left 3 by 3 side of P into K and R via QR factorization:

$$K=\begin{bmatrix}5.55\times10^{2}&0.00&9.20\times10^{1}\\0.00&1.41\times10^{2}&4.62\times10^{2}\\0.00&0.00&1.00\end{bmatrix}R=\begin{bmatrix}-3.57\times10^{2}&2.44&-1.66\times10^{5}\\3.24\times10^{2}&1.29\times10^{1}&-7.65\times10^{5}\\8.94\times10^{-2}&4.97\times10^{2}&-2.33\times10^{5}\end{bmatrix}$$

Solving for t:

$$t=K^{-1}\begin{bmatrix}p_{14}\\p_{24}\\p_{34}\end{bmatrix}=\begin{bmatrix}1.80\times10^{-3}&0.00&-1.66\times10^{-1}\\0.00&7.11\times10^{-3}&-3.29\times10^{0}\\0.00&0.00&1.00\end{bmatrix}\begin{bmatrix}-7.92\times10^{-1}\\-6.10\times10^{-1}\\-8.12\times10^{-4}\end{bmatrix}=\begin{bmatrix}-1.29\times10^{-3}\\-1.67\times10^{-3}\\-8.12\times10^{-4}\end{bmatrix}$$

Thus, the intrinsic and extrinsic matrices are:

$$intrinsic = \begin{bmatrix} 5.55 \times 10^2 & 0.00 & 9.20 \times 10^1 & 0 \\ 0.00 & 1.41 \times 10^2 & 4.62 \times 10^2 & 0 \\ 0.00 & 0.00 & 1.00 & 0 \end{bmatrix} \quad extrinsic = \begin{bmatrix} -3.57 \times 10^2 & 2.44 & -1.66 \times 10^5 & -1.29 \times 10^{-3} \\ 3.24 \times 10^2 & 1.29 \times 10^1 & -7.65 \times 10^5 & -1.67 \times 10^{-3} \\ 8.94 \times 10^{-2} & 4.97 \times 10^2 & -2.33 \times 10^5 & -8.12 \times 10^{-4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The last part of the question asks for the degree of rotation about each axis.

With Euler angle formulas they can be achieved:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \begin{aligned} \theta_x &= \text{atan2}(r_{32}, r_{33}) \\ \theta_y &= \text{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \theta_z &= \text{atan2}(r_{21}, r_{11}) \end{aligned}$$

$$R = \begin{bmatrix} -3.57 \times 10^2 & 2.44 & -1.66 \times 10^5 \\ 3.24 \times 10^2 & 1.29 \times 10^1 & -7.65 \times 10^5 \\ 8.94 \times 10^{-2} & 4.97 \times 10^2 & -2.33 \times 10^5 \end{bmatrix}$$

Thus:

$$\theta_x = 179.87°, \quad \theta_y = -2° \times 10^{-5} \approx 0°, \quad \theta_z = 137.73°$$

---

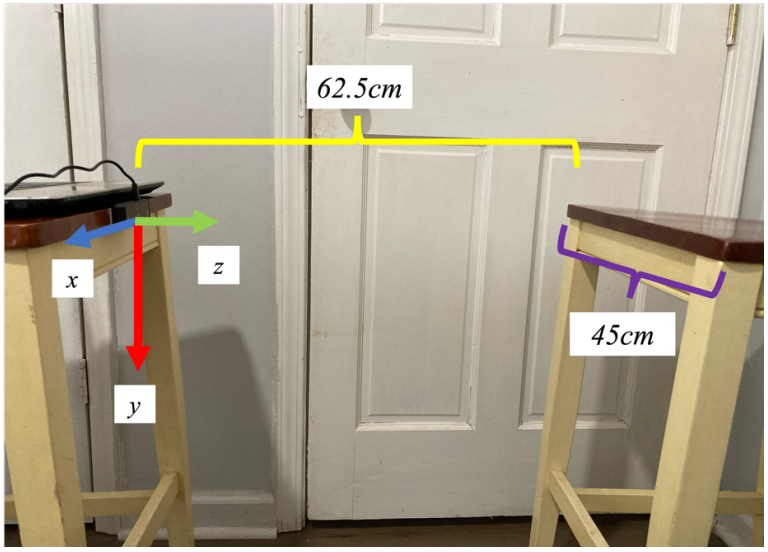
3. Write a script to find the real world dimensions (e.g. diameter of a ball, side length of a cube) of an object using perspective projection equations. Validate using an experiment where you image an object using your camera from a specific1 distance (choose any distance but ensure you are able to measure it accurately) between the object and camera.

In order to do this, we need the camera coordinates of the two points at the start and end of the object. Now, perspective projection is a one-way equation, meaning, infinite many points in camera coordinates can be mapped to a point in the image with pixel coordinates. However, by knowing the distance between the object and the camera, i.e., z_c this is easily solvable by:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} = K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

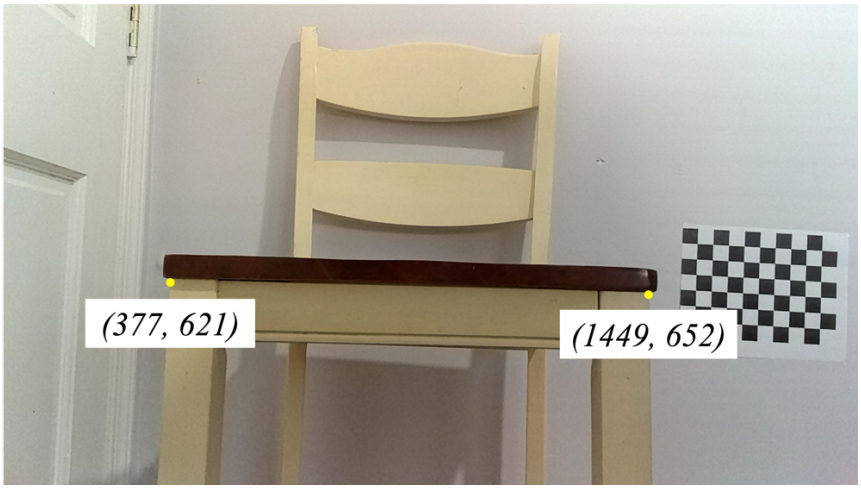
So basically, the homogeneous camera coordinates are obtained by multiplying the inverse of calibration matrix with the homogeneous pixel coordinates, and further scaled by the factor of z_c. The length of the object is then achievable by finding the Euclidean distance between the ends.

Before writing the script, I will manually solve this for an example.



I am trying to get camera coordinates of the end of the sides of the chair's seat, and then calculate the Euclidean distance between them (45cm). The z_c coordinate of them is 62.5cm as the camera is placed within this distance from the chair.

First, I will get the pixel coordinates of the ends of the chair's seat side.



Now I will multiply each one's homogeneous coordinates with the inverse of the calibration matrix obtained in question one to get homogeneous camera coordinates, and get the real camera coordinates of them via scaling by 62.5:

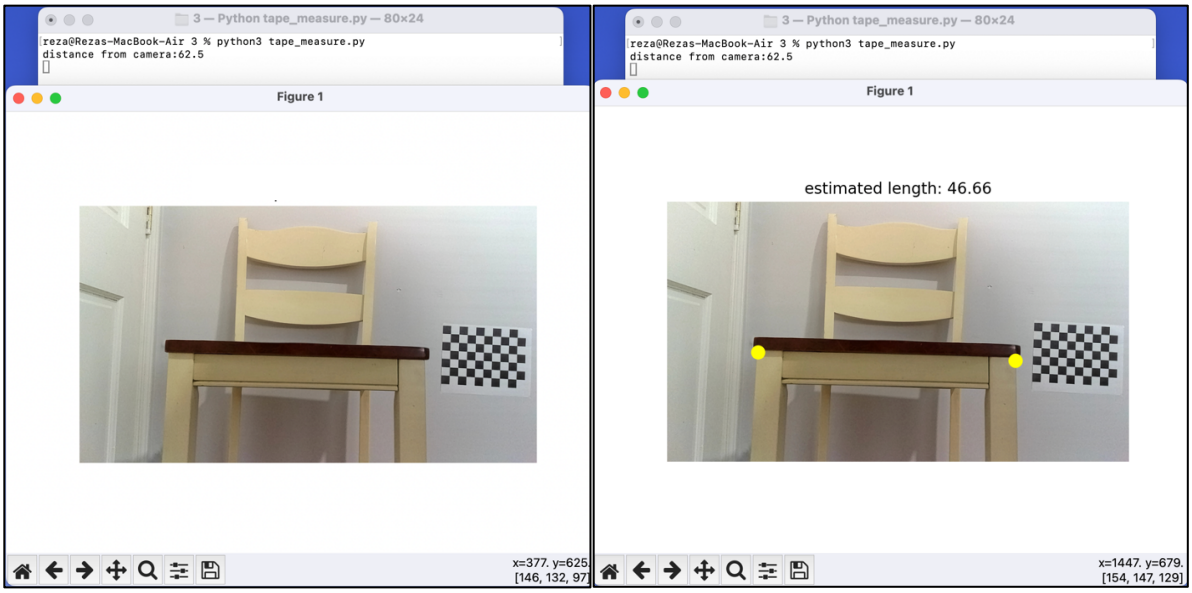
$$\begin{bmatrix} x_{c_1} \\ y_{c_1} \\ z_{c_1} \end{bmatrix} = \begin{bmatrix} 6.97 \times 10^{-4} & 0.00 & -6.62 \times 10^{-1} \\ 0.00 & 6.99 \times 10^{-4} & -3.78 \times 10^{-1} \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \cdot \begin{bmatrix} 377 \\ 621 \\ 1 \end{bmatrix} \times 62.5 = \begin{bmatrix} -0.399 \\ 0.056 \\ 1 \end{bmatrix} \times 62.5 = \begin{bmatrix} -24.96 \\ 3.48 \\ 62.5 \end{bmatrix}$$

$$\begin{bmatrix} x_{c_2} \\ y_{c_2} \\ z_{c_2} \end{bmatrix} = \begin{bmatrix} 6.97 \times 10^{-4} & 0.00 & -6.62 \times 10^{-1} \\ 0.00 & 6.99 \times 10^{-4} & -3.78 \times 10^{-1} \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \cdot \begin{bmatrix} 1449 \\ 652 \\ 1 \end{bmatrix} \times 62.5 = \begin{bmatrix} 0.348 \\ 0.077 \\ 1 \end{bmatrix} \times 62.5 = \begin{bmatrix} 21.75 \\ 4.83 \\ 62.5 \end{bmatrix}$$

Now calculating the distance between them:

$$d = \sqrt{(-24.96 - 21.75)^2 + (3.48 - 4.83)^2 + (62.5 - 62.5)^2} = 46.7 \approx 45$$

Here's the result of the code that will ask the user for distance from the camera and to click on the object's ends and return the length of the object:



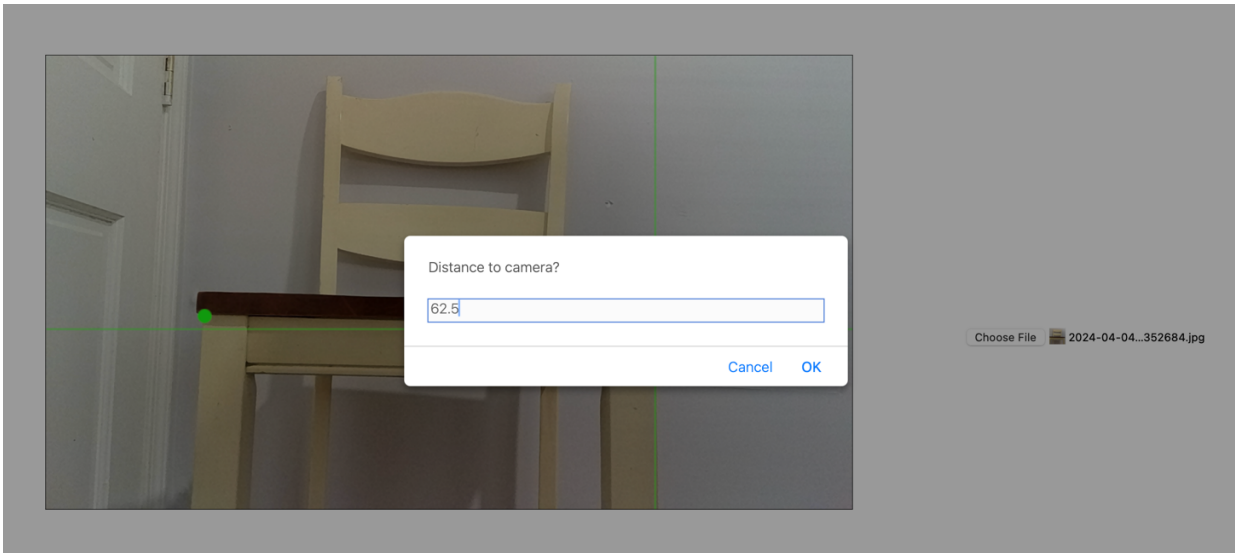
And the script is (also available in the attachments and on the GitHub repo):

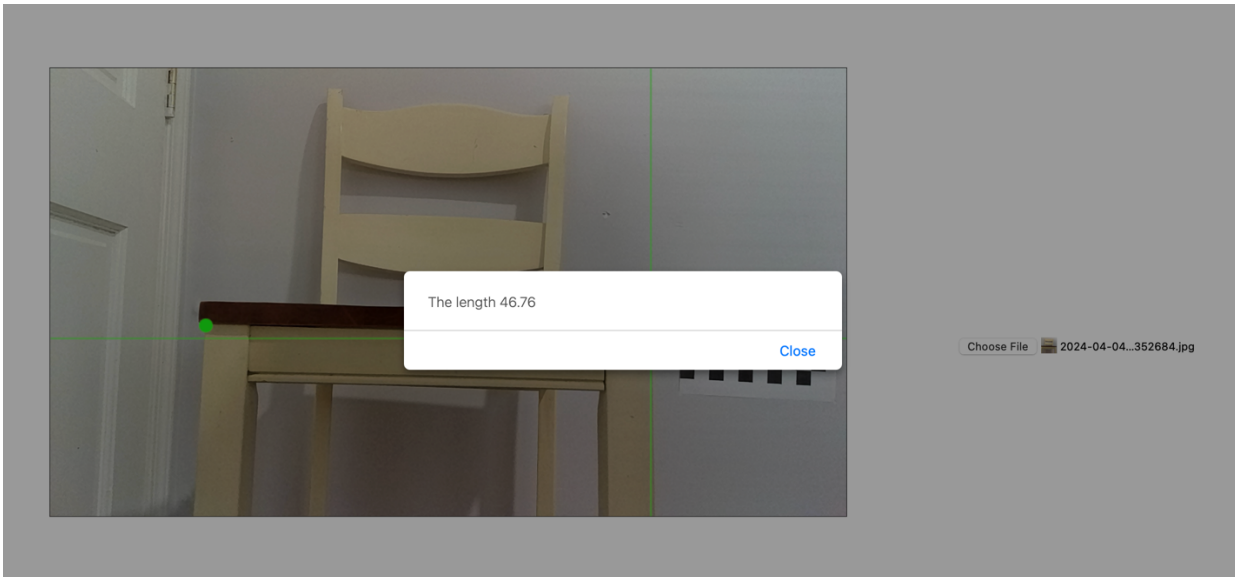
```python
import numpy as np
import matplotlib.pyplot as plt

cid = None
click_count = 0
uv1, uv2 = None, None
K_inv = np.linalg.inv(
    np.array([
        [1434.41, 0, 949.77],
        [0, 1430.68, 541.41],
        [0, 0, 1]])
)


def onclick(event):
    global uv1, uv2, click_count
    homogeneous_pixel_coords = np.array([[event.xdata], [event.ydata], [1]])
    if click_count == 0:
        uv1 = homogeneous_pixel_coords
    else:
        uv2 = homogeneous_pixel_coords
        plt.close()
```

```
    click_count += 1


def main():
    global uv1, uv2
    image = plt.imread('image_path')
    zc = input('distance from camera:')
    zc = float(zc)
    fig, ax = plt.subplots()
    ax.imshow(image)
    ax.set_title('Click to print coordinates')
    ax.axis('off')
    fig.canvas.mpl_connect('button_press_event', onclick)
    plt.show()
    plt.axis('off')
    plt.imshow(image)
    plt.scatter(uv1[0,0], uv1[1,0], color='yellow', s=100, zorder=2)
    plt.scatter(uv2[0,0], uv2[1,0], color='yellow', s=100, zorder=2)
    xyz1 = K_inv.dot(uv1) * zc
    xyz2 = K_inv.dot(uv2) * zc
    length = np.linalg.norm(xyz1 - xyz2)
    plt.title(f'estimated length: {length:.2f}')
    plt.show()


if __name__ == '__main__':
    main()
```

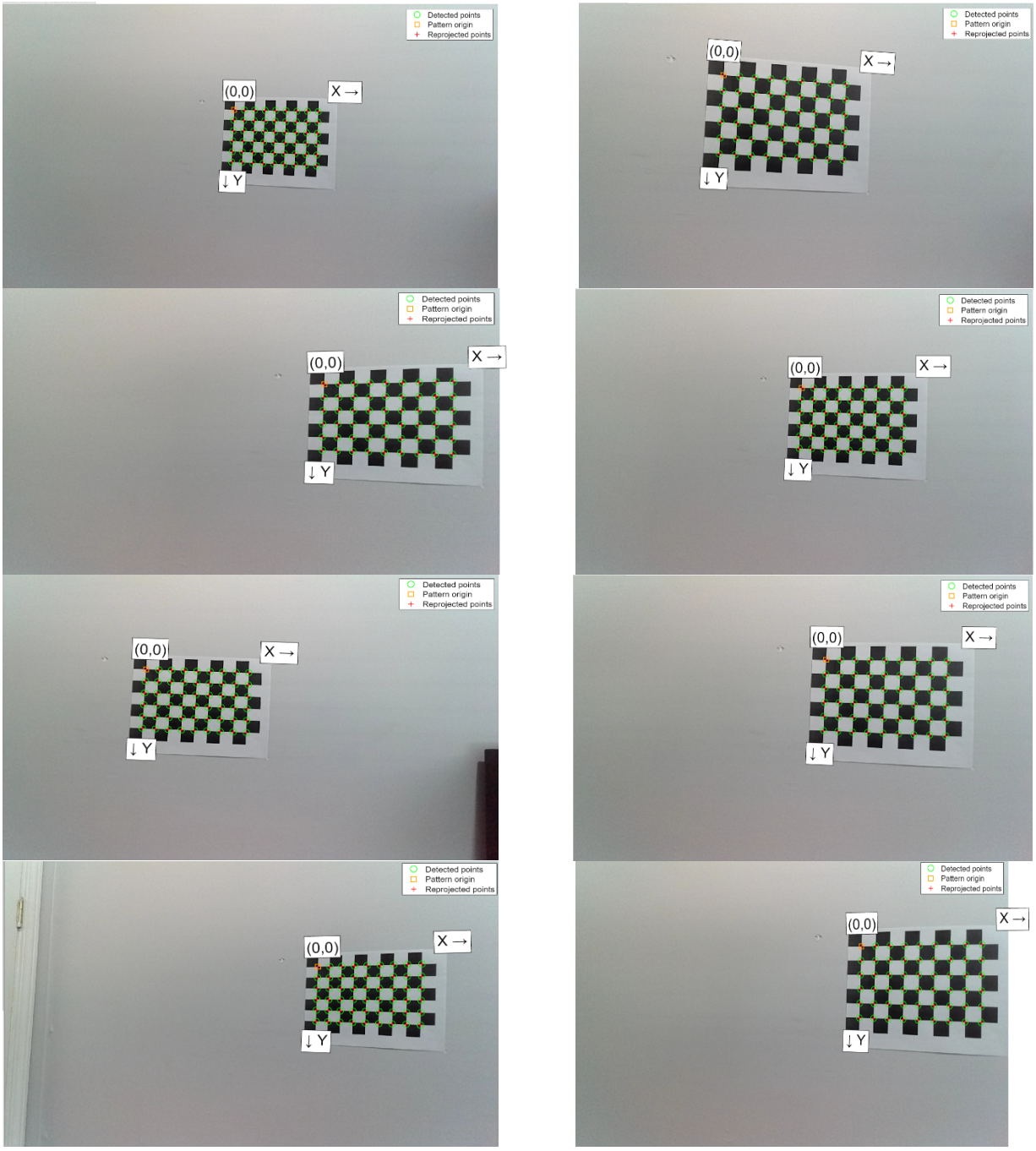
4. Write an application – must run as a Web application on a browser and be OS agnostic – that implements the solution for problem (3) [An application that can compute real-world dimensions of an object in view]. Make justifiable assumptions (e.g. points of interest on the object can be found by clicking on the view or touching on the screen).

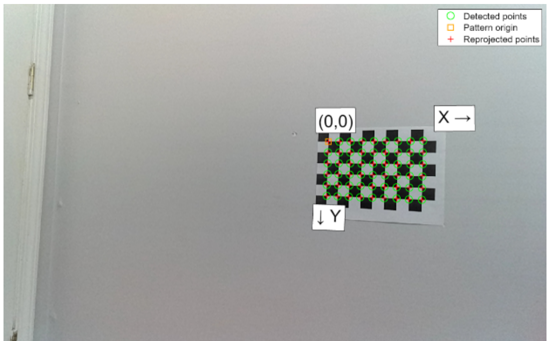
Here is the footage of the web-application. The code is attached, also available on the GitHub repo, and you can try the application at https://rezmansouri.github.io/csc8830.

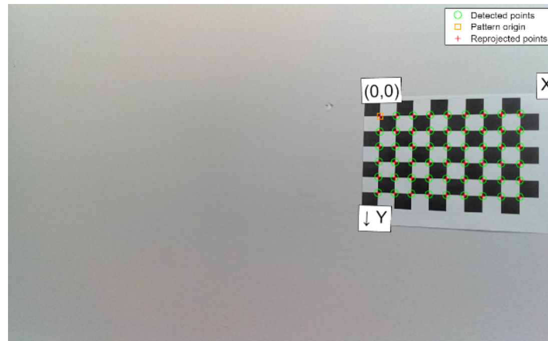


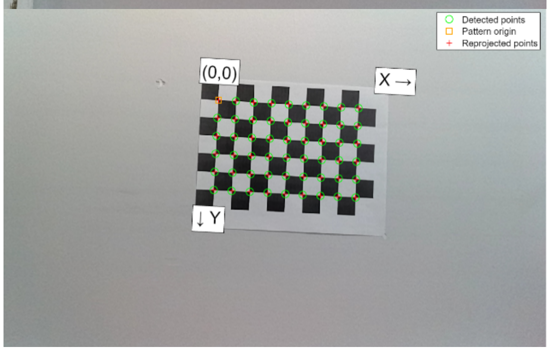


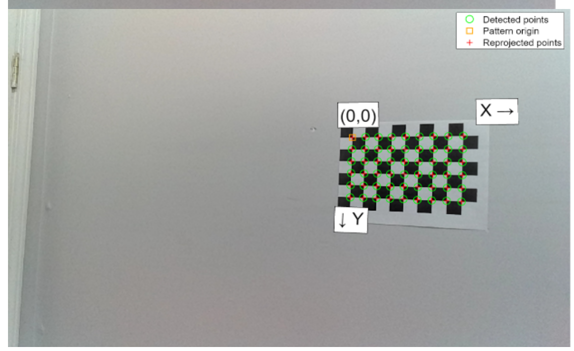
APPENDIX I: Calibration Footage

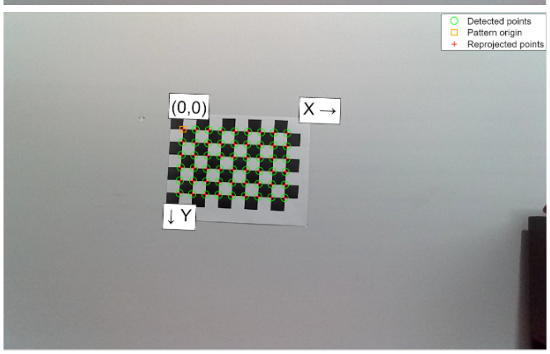


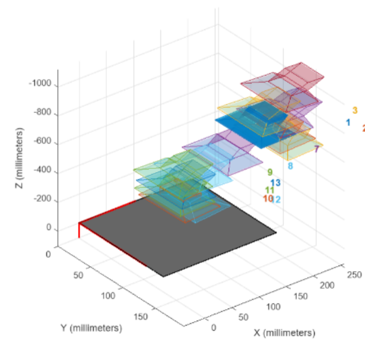

Detected points
Pattern origin
Reprojected points
(0,0)
X →
↓ Y

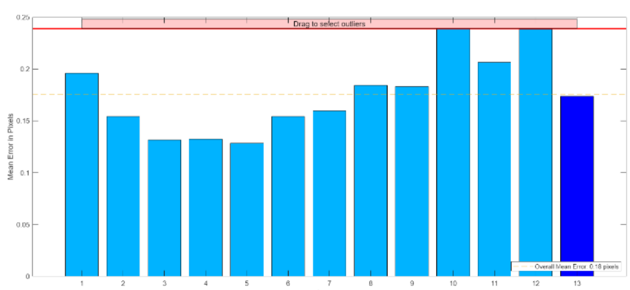

Detected points
Pattern origin
Reprojected points
(0,0)
X →
↓ Y


Detected points
Pattern origin
Reprojected points
(0,0)
X →
↓ Y


Detected points
Pattern origin
Reprojected points
(0,0)
X →
↓ Y


Detected points
Pattern origin
Reprojected points
(0,0)
X →
↓ Y


Z (millimeters)
Y (millimeters)
X (millimeters)


Drag to select outliers
Mean Error in Pixels
Images