Computer Vision 3D (67542): Exercise 2

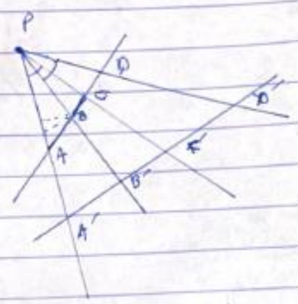
By: Jeremie Sasson, ID: 336319470

Amit Halbreich, ID: 208917393

Question 1

Section (A) - Proof

Cross ratio invariance under projective transformation



with AB: distance between A and B

- $\text{cross}(ABCD) = \dfrac{AB \cdot CD}{AD \cdot CB}$

Let's remember the law of sines in a triangle PAB:

$$\frac{\sin\angle PBA}{PA} = \frac{\sin\angle APB}{AB}$$

So, applying this formula to triangles APB, PCD, APD, PCB

$$\text{cross}(ABCD) = \frac{\dfrac{\sin APB \; PA}{\sin PBA} \cdot \dfrac{\sin CPD \; PC}{\sin PDC}}{\dfrac{\sin APD \; PA}{\sin PDA} \cdot \dfrac{\sin BPC \; PC}{\sin PBC}}$$

- $\left(\begin{array}{ll} PBA = 180 - PBC & \text{so } \boxed{\sin PBA = \sin PBC} \\ PDA = PDC & \text{so } \sin PDA = \sin PDC \end{array} \right)$

$$= \frac{\sin APB \; \sin CPD}{\sin APD \; \sin BPC} \; \frac{PA \;\; PC}{PA \;\; PC} \; (=1)$$

$$= \frac{\sin A'PB' \; \sin C'PD'}{\sin A'PD' \; \sin B'PC'} \times \frac{PA' \;\; PC'}{PA' \;\; PC'}$$

again law of sines

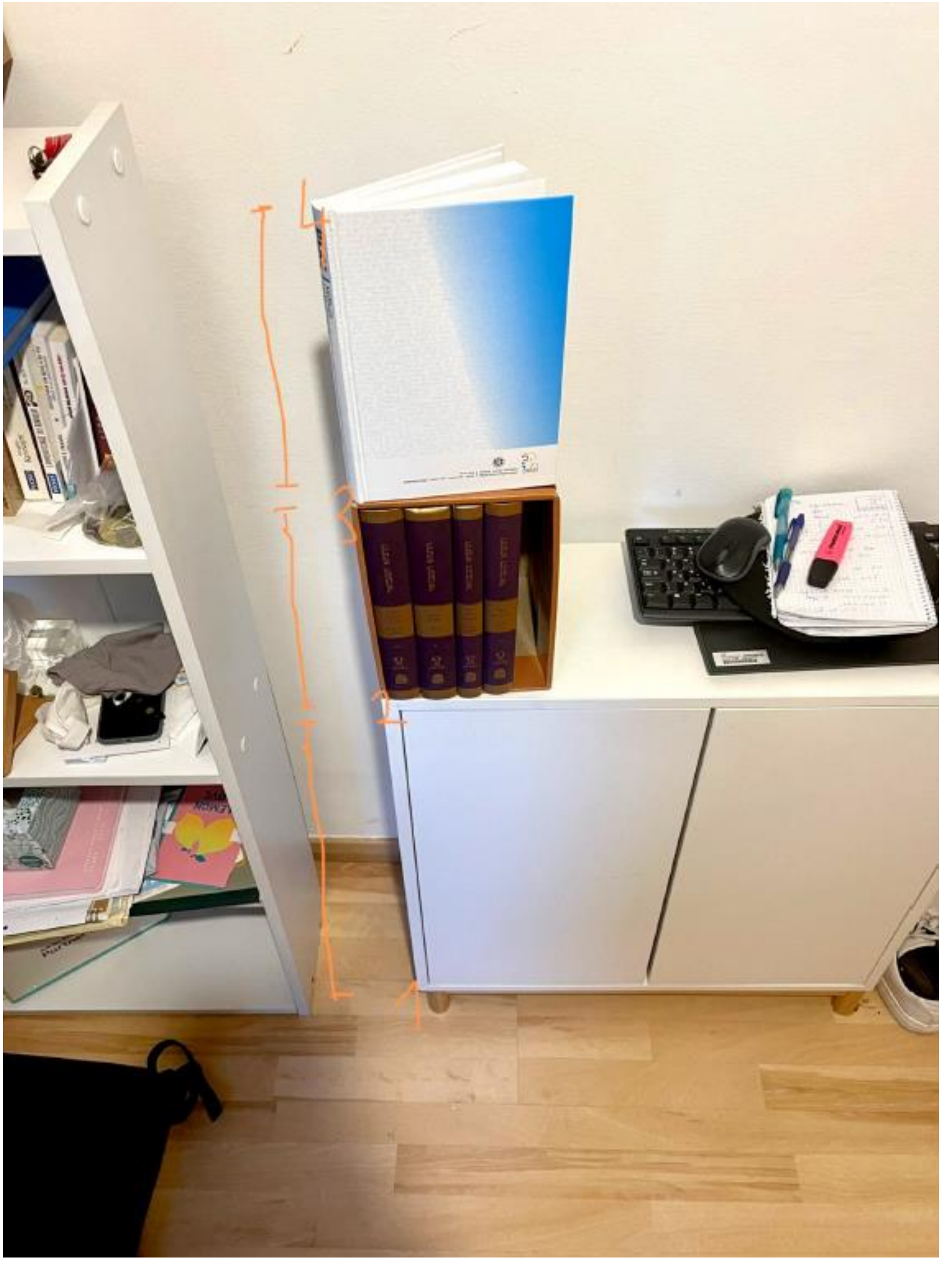
$$= \frac{\sin PB'A' \times A'B'}{\sin PD'A' \;\; A'D'} \; \frac{\sin PD'C' \times C'D'}{\sin PB'C' \times B'C'}$$

equal sinus

$$= \frac{A'B' \;\; C'D'}{A'D' \;\; B'C'}$$

SECTION (B)

We used 4 known points P_1, P_2, P_3, P_4 in the picture corresponding to points P_1', P_2', P_3', P_4' and used them to measured distances in the real world $P_3'P_4' = 24.5\ [cm]$, $P_2'P_4' = 50[cm]$ in order to find the height of the dresser denoted by $P_1'P_2'$. The image we used:



CALCULATION OF DRESSER HEIGHT

On the picture

$P_3P_4 = 2,2$ cm

$P_3P_2 = 1,4$ cm

$P_2P_1 = 2,1$ cm

In real world.

$P_3'P_4' = 24,5$ cm

$P_2P_4' = 50$ cm.

Let's find $P_1'P_2'$ (real world)

$$\frac{(P_1'P_3')(P_2'P_4')}{(P_2'P_3')(P_1'P_4')} = \frac{(P_1P_3)(P_2P_4)}{(P_2P_3)(P_1P_4)}$$

$$(P_1'P_2' + P_2'P_3') = \frac{3.5 \times 3.6}{1.4 \times 5.7} \times \frac{(P_1'P_2' + P_2'P_4')\, P_2'P_3'}{P_2'P_4'}$$

$$P_1'P_2' + 25.5 = \frac{3.5 \times 3.6}{1.4 \times 5.7} \times \frac{(P_1'P_2' + 50)\, 25.5}{50}$$

$$P_1'P_2' = (P_1'P_2' + 50) \times 0.805 - 25.5$$

$$0.194 \times P_1'P_2' = 40.26 - 25.5$$

$$P_1'P_2' = 76 \text{ cm}$$

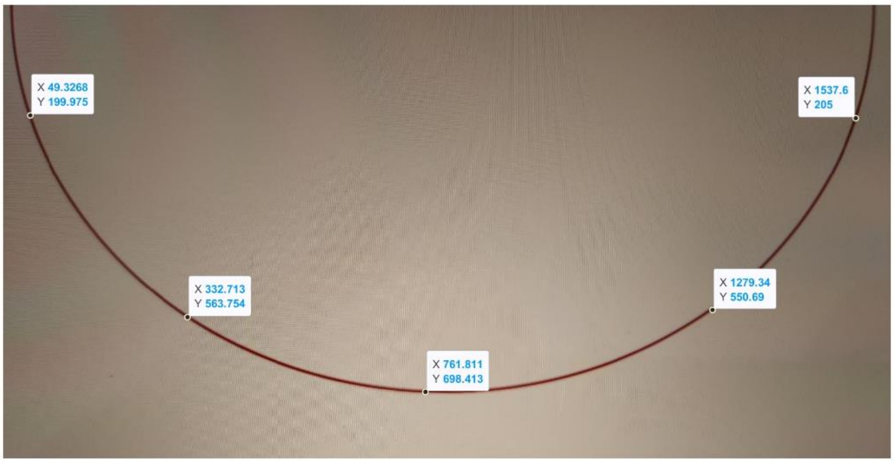
It is indeed the height of dresser.

The height of the dresser we got using Cross-Ratio formula is indeed very close to the measured real height of the dresser which makes a lot of sense.

$$Dresser\ Height = P_1'P_2 = 76[cm]$$

QUESTION 2

The image we used with the following coordinates (x, y) relative to pixels in the image:



We verified that the following coordinates indeed are on a parabola using this function:

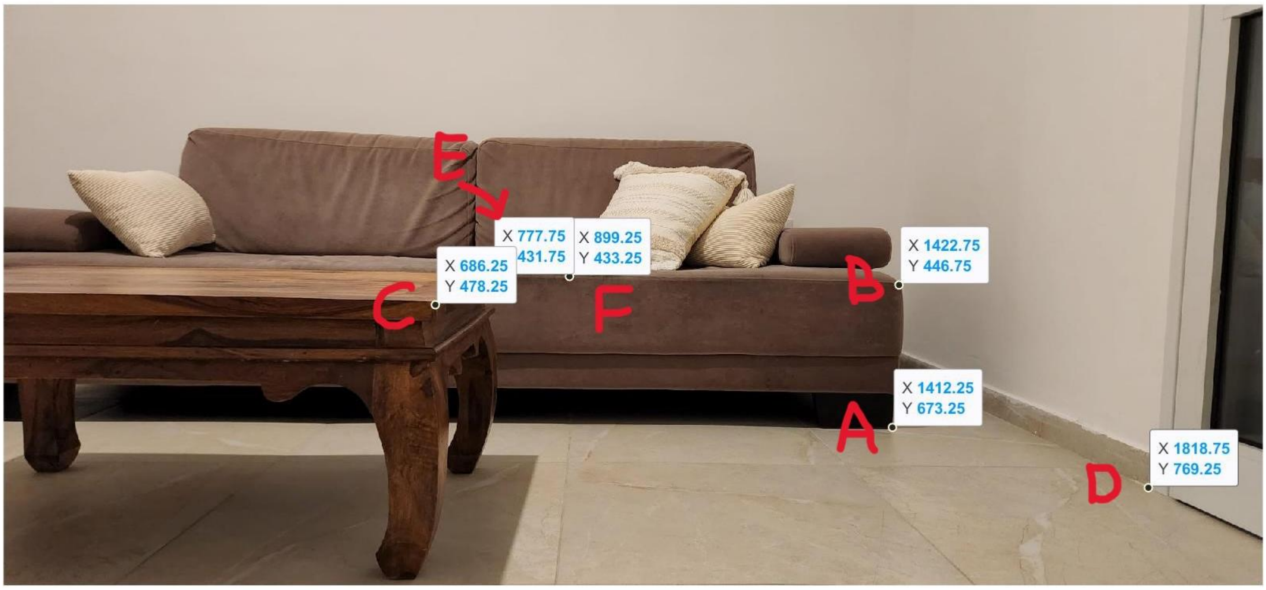
```
def is_parabola(points, epsilon=1e-5):
    # Fit a conic to the extracted points
    A = np.vstack(
        [points[:, 0] ** 2, points[:, 0] * points[:, 1], points[:, 1]
** 2,
         points[:, 0], points[:, 1], np.ones(points.shape[0])]).T
    _, _, V = np.linalg.svd(A)
    coefficients = V[-1, :]
    A, B, C, D, E, F = coefficients

    # Compute the discriminant
    discriminant = B ** 2 - 4 * A * C
    print("The discriminant is:")
    print(discriminant)

    # Check if the discriminant is close to zero within epsilon
value,
    # indicating a parabola
    if np.isclose(discriminant, 0, rtol=0, atol=epsilon):
        return True
    else:
        return False

points = np.array([[49.3268, 199.975], [332.713, 563.754], [761.811,
                                                             698.413],
          [1279.34, 550.69], [1537.6, 205.0]])

result = is_parabola(points)
if result:
    print("The imaged curve is a parabola.")
else:
    print("The imaged curve is not a parabola.")
```

We got the following result which is very close to the value of $\Delta = 0$ where Δ denotes the discriminant:

```
The discriminant is:
-4.276746268339605e-09
The imaged curve is a parabola.
```

QUESTION 3

The image we used:



Define 6 points in the projective homogenous space in 3D world coordinates relative to a selected fixed origin $O = (0,0,0,1)$, coordinates are measured in [cm]:

$$A = (81.5, 0, 214, 1)$$

$$B = (81.5,37.5, 214, 1)$$

$$C = (-18.5, 40, 118.5, 1)$$

$$D = (115,0, 166.2\ , 1)$$

$$E = (-16, 40, 178.5, 1)$$

$$F = (0, 37.5, 214, 1)$$

Find the 6 chosen coordinates in a 2D image, coordinates in homogeneous space:

$$A' = (1412.25, 673.25, 1)$$

$$B' = (1422.75, 437.25, 1)$$

$$C' = (686.25, 478.25, 1)$$

$$D' = (1818.75, 769.25, 1)$$

$$E' = (777.75, 431.75, 1)$$

$$F' = (899.25, 433.25, 1)$$

\

Now we ran DLT Algorithm we defined, in order to find the Matrix representing the Camera:

$$P = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix}$$

And from here we got the camera's actual location by finding the kernel of the transform matrix P.

The code we used defining DLT algorithm:

```
def find_camera_location(points_3d, points_2d):
    """
    Find the camera's location (extrinsic parameters) using DLT
algorithm.

    Args:
        points_3d (np.ndarray): Array of 3D points in homogeneous
coordinates, shape (n, 4)
        points_2d (np.ndarray): Array of 2D points in homogeneous
coordinates, shape (n, 3)

    Returns:
        np.ndarray: Array representing the camera's location in
homogenous
        coordinates.
        shape (1, 4)
    """
    # Construct the A matrix
    num_points = points_3d.shape[0]
    A = np.zeros((2 * num_points, 12))
    for i in range(num_points):
        X, Y, Z,_ = points_3d[i]
        u, v, w = points_2d[i]
        A[2 * i] = [-X, -Y, -Z, -1, 0, 0, 0, 0, u * X, u * Y, u * Z,
u]
        A[2 * i + 1] = [0, 0, 0, 0, -X, -Y, -Z, -1, v * X, v * Y, v *
Z, v]

    # Find the camera matrix, solving the system Ax=0, the result is
the last vector column of V
    u, s, vh = np.linalg.svd(A)
    P = vh[-1].reshape((3, 4))

    # Extract the camera center, again we solve Px=0, the result is
the last vector column of V
    _, _, Vh = svd(P)  # s is a vector, use  diag(s) to get the
matrix sigma
    camera_location  = Vh[-1]

    # Casting to homogeneous coordinates
    return camera_location/camera_location[3]
```

```python
    # Define the points in 3D homogeneous space
    points_3d = np.array([
        [81.5, 0, 214, 1],
        [81.5, 37.5, 214, 1],
        [-18.5, 40, 118.5, 1],
        [115, 0, 166.2, 1],
        [-16, 40, 178.5, 1],
        [0, 37.5, 214, 1]
    ])

    # Define the points in 2D image space
    points_2d = np.array([
        [1412.25, 673.25, 1],
        [1422.75, 437.25, 1],
        [686.25, 478.25, 1],
        [1818.75, 769.25, 1],
        [777.75, 431.75, 1],
        [899.25, 433.25, 1]
    ])
#
    # Find the camera's location (extrinsic parameters)
    camera_location = find_camera_location(points_3d, points_2d)
    # Print the camera's location
    print("Camera's Location in Homogenous Coordinates (X,Y,Z,1):")
    print(camera_location)
```

We got the result for the camera's location:

```
Camera's Location in Homogenous Coordinates (X,Y,Z,1):
[ 1.87144785 52.07177235  0.51244873  1.        ]
```

$$Camera's\ Location = [x, y, z, 1] = [1.87144785\ \ ,52.07177235, 0.51244873,\ \ 1.0]$$

If we compare it to the actual estimated location we measured originally, we got:

$$Camera's\ Measured\ Location =$$

$$[x, y, z, 1] = [0\ , 50, 0, 1]$$

As expected, we indeed got a close solution to the original camera's location we measured, which indicates the correctness of our DLT algorithm code.

We have taken into account small measurement faults made and camera positioning that was slightly with angle and not facing exactly towards the Z axis positive direction (front of the camera was with a little angle) and still we got good results.