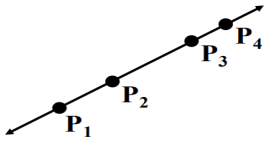
Question 1

Cross ratio

(a) Prove that the cross-ratio is invariant under the projective transformation (use homogeneous coordinates):
cross-ratio means that a straight line under a projective transformation remains straight.
And it's supposed to be invariant accross projective transformations:



$$\frac{\|\mathbf{P}_3-\mathbf{P}_1\|\,\|\mathbf{P}_4-\mathbf{P}_2\|}{\|\mathbf{P}_3-\mathbf{P}_2\|\,\|\mathbf{P}_4-\mathbf{P}_1\|} \qquad \mathbf{P}_i=\begin{bmatrix}X_i\\Y_i\\Z_i\\1\end{bmatrix}$$

Let A, B, C, D be 4 collinear points (on the same line) and let Hbe a projective transformation.
We know from the proparty of projective transformations that they preserve straightness and so $HA, HB, HC, HD = A', B', C', D'$
are on the same line.
Lets denote another point $O = \begin{pmatrix} o_1 & o_2 & 1 \end{pmatrix}$ and $HO = O'$ a point that is not collinear with A', B', C', D'
Then we get that $(A', B'; C', D') = \frac{[O',A',C']\cdot[O',B',D']}{[O',B',C']\cdot[O',A',D']}$ from Menelaus' theorem
It also holds that

$$[O', A', C'] = \begin{vmatrix} O'_1 & A'_1 & C'_1 \\ O'_2 & A'_2 & C'_2 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} O'_1 & A'_1 - O'_1 & C'_1 - O'_1 \\ O'_2 & A'_2 - O'_2 & C'_2 - O'_2 \\ 1 & 0 & 0 \end{vmatrix} = 1 \cdot \begin{vmatrix} A'_1 - O'_1 & C'_1 - O'_1 \\ A'_2 - O'_2 & C'_2 - O'_2 \end{vmatrix} = [HOHA, HOHC] = det\,(HOH) \cdot [A, C]$$

It can be done to $[O', B', D'], [O', B', C'], [O', A', D']$ as well:
So $(A', B'; C', D') = \frac{[O',A',C']\cdot[O',B',D']}{[O',B',C']\cdot[O',A',D']} = \frac{det(HOH)^2[A,C]\cdot[B,D]}{det(HOH)^2[B,C]\cdot[A,D]} = \frac{[A,C]\cdot[B,D]}{[B,C]\cdot[A,D]} = (A, B; C, D)$

(b) Measure size of table

$d = ab \times a'b'$ (a point in 'infinity')

$T = table\ size$

$S = size\ of\ the\ book$

I used this code for calculations:

$\frac{\|c-a\|\|d-b\|}{\|b-a\|\|d-c\|} = \frac{T}{S}$

$\Rightarrow S \cdot \frac{\|c-a\|\|d-b\|}{\|b-a\|\|d-c\|} = T$

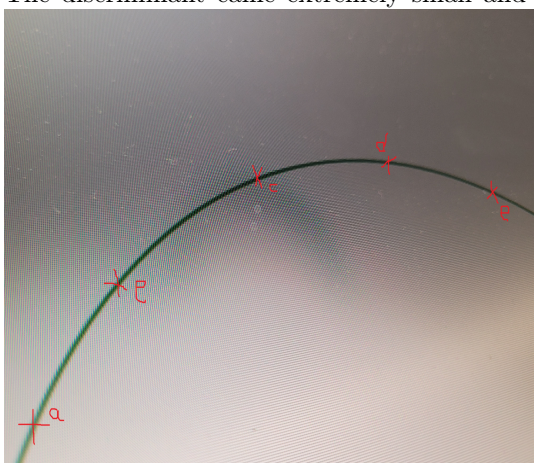
```
REAL_DIST_AB = 25.5
def run_compute_table_size():
    a = np.array([2724, 2863, 1])
    b = np.array([1310, 1743, 1])
    a_tag = np.array([3986, 1792, 1])
    b_tag = np.array([2363, 1259, 1])
    c = np.array([166, 857, 1])
    line_ab = do_cross(a, b)
    line_atbt = do_cross(a_tag, b_tag)
    d = do_cross(line_atbt, line_ab)
    T = dist(c, a) * dist(d, b)/(dist(b, a) * dist(d, c)) * REAL_DIST_AB
    print(f"size of table is {np.round(T, 1)} cm, vs real size of 110 cm")
def dist(x, y):
    return np.linalg.norm(x[:-1] - y[:-1])
def do_cross(a, b):
    line_ab = np.cross(a, b).astype(float)
    line_ab /= line_ab[-1]
    return line_ab

dist()
main
"C:\Users\anita\OneDrive\Desktop\master\vision 3D\ex2\venv\Scripts\python.exe" "C:
size of table is 114.2 cm, vs real size of 110 cm
```

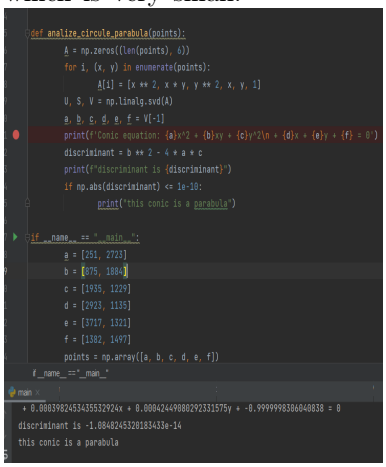
The result I got was 3.5 percent of the real distance.

Question 2

I found 6 points on a circle that was picured from the middle.
fit the points to a conic and found the conic parameters using SVD and the following code:
The discriminant came extremely small and this is why I can say that it was indeed prodected to the image as a parabula.



I used this code: created the conic points matrix A, and then found the nullspace of this matrix the represented the conic variables.
because the data is noisy there was neber going to be a perfect parabula, and therefore thee discriminant was never going to be exacly 0
and so I wanted to be smaller then a certain ε in order to state that this is indeed a parabula. And iit came out $1.08 \cdot 10^{-14}$ which is very small.



```
def analize_circule_parabula(points):
    A = np.zeros((len(points), 6))
    for i, (x, y) in enumerate(points):
        A[i] = [x ** 2, x * y, y ** 2, x, y, 1]
    U, S, V = np.linalg.svd(A)
    a, b, c, d, e, f = V[-1]
    print(f'Conic equation: {a}x^2 + {b}xy + {c}y^2\n + {d}x + {e}y + {f} = 0')
    discriminant = b ** 2 - 4 * a * c
    print(f"discriminant is {discriminant}")
    if np.abs(discriminant) <= 1e-10:
        print('this conic is a parabula')

if __name__ == "__main__":
    a = [251, 2723]
    b = [875, 1884]
    c = [1935, 1229]
    d = [2923, 1135]
    e = [3717, 1321]
    f = [1382, 1497]
    points = np.array([a, b, c, d, e, f])
```

```
if __name__=="__main__"
main
  + 0.08839824534355532924x + 0.009424490880292331575y + -0.9999998306040838 = 0
discriminant is -1.0840245328183433e-14
this conic is a parabula
```

Question 3

Extracting camera location:

I used the following points:

the real world coordinates are:

a = (60, 0, 0, 1) b = (0, 0, 0, 1) c = (60, 60, 0, 1)

d = (0, 60, 0, 1) e = (0, 105, 10, 1) f = (60, 105, 10, 1)

And the image coordinates are:

A = (1862, 2570, 1) B = (162, 2097, 1) C = (1988, 1511, 1)

D = (880, 1379, 1) E = (1137, 984, 1) F = (2049, 1022, 1)

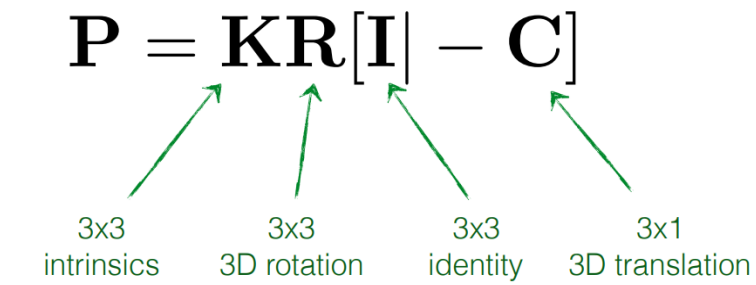
And use the following points to find the elements of the camera matrix using the following formula:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

I found the camera matrix using this code:
basically it's bulding the big matrix from the above formula, and then finds it's nullspace using the last column of the SVD decomposition.

```python
def find_fundamental_matrix(world_points, image_points):
        n = world_points.shape[0]
        # Construct the matrix A
        A = np.zeros((2 * n, 12))
        for i in range(n):
                x, y, z = world_points[i][:-1]
                u, v = image_points[i][:-1]
                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]
        # Solve for the nullspace of A
        _, _, V = np.linalg.svd(A)
        P = V[-1].reshape(3, 4)
        P /= P[-1, -1]
        return P
```

Then I found the C component:



Using this code:

```python
# Compute the camera center
C = -np.linalg.inv(P[:, :3]) @ P[:, 3]
```

The result that I got was:

```
camera location is [ 59.06059393 -36.792746    58.83892228] cm
```

which make sense as b=(0,0,0) and a=(60, 0, 0)
So it's in a's x axis, negative y and z around 60 cm.