```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

Q1 Estimating door size

```
def homography_transform(src_pts: np.ndarray, dest_pts: np.ndarray):
    homography_matrix = np.zeros((2*len(src_pts), 9))
    for i in range(len(src_pts)):
        x, y = src_pts[i]
        x_dash, y_dash = dest_pts[i]
        homography_matrix[2*i] = [x, y, 1, 0, 0, 0, -x_dash*x, -x_dash*y, -x_dash]
        homography_matrix[2*i+1] = [0, 0, 0, x, y, 1, -y_dash*x, -y_dash*y, -y_dash

    eigen_value, eigen_vectors = np.linalg.eig(homography_matrix.T @ homography_mat
    min_eigen_val_index = np.argmin(eigen_value)
    return eigen_vectors[:, min_eigen_val_index].reshape(3,3)

test_image = plt.imread('Assignment4/my_image.jpg')

door_corners = np.float32([[308, 648],
                          [424, 2265],
                          [3813, 777],
                          [3771, 1907]])

object_corners = np.float32([[2103, 1361],
                            [2107, 1483],
                            [2374, 1349],
                            [2371, 1468]])

#in mm
object_real_height, object_real_width = 165, 72
door_destination = np.float32([[0,0],[0, 800], [2000, 0],[2000, 800]])
```

```
homography_matrix = homography_transform(door_corners, door_destination)
transformed_corners = cv2.perspectiveTransform(object_corners.reshape(1,-1,2), homo
w = np.linalg.norm(transformed_corners[0][0] - transformed_corners[0][1])
h = np.linalg.norm(transformed_corners[0][0] - transformed_corners[0][2])

print(f"Object relative width: {w} px, Object relative height: {h} px")

door_abs_width = 800 * w / object_real_width
door_abs_height = 2000 * h / object_real_height
print(f"Door width: {door_abs_width}, Door height: {door_abs_height}")
```

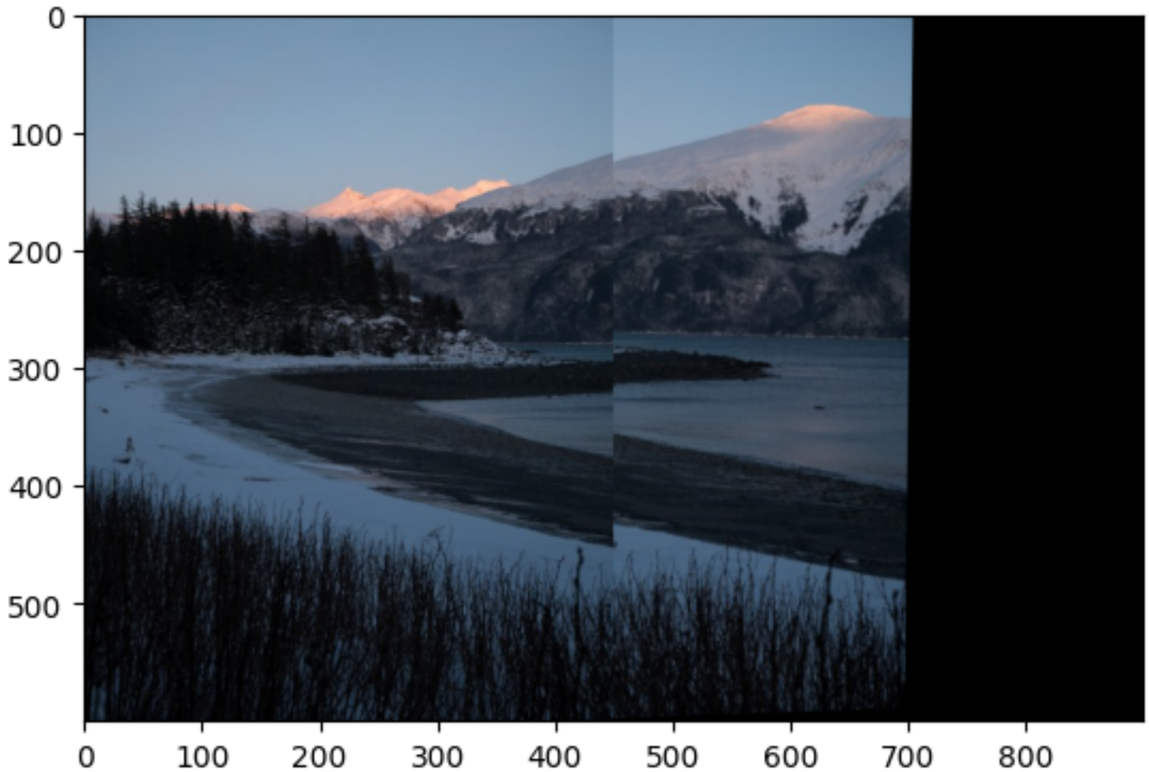
```
Object relative width: 71.34793853759766 px, Object relative height: 158.2504
Door width: 792.754872639974, Door height: 1918.1875517874053
```

Q2 Landscape stitching

```
def extract_feature(image: np.ndarray):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    sift = cv2.SIFT_create()
    gray_image = cv2.normalize(gray_image, None, 0, 255, cv2.NORM_MINMAX).astype(
    keypoints, descriptors = sift.detectAndCompute(gray_image, None)

    return keypoints, descriptors

def match_features(descriptors1: np.ndarray, descriptors2: np.ndarray, threshold:
    """ Matches the features using the ratio of euclidean distance between the cl
    """
    matches = []
    for i in range(descriptors1.shape[0]):
        distances = np.linalg.norm(descriptors2 - descriptors1[i], axis=1)
        closest, second_closest = np.argsort(distances)[:2]
        ratio = distances[closest] / distances[second_closest]
        if ratio < threshold:
            matches.append((i, closest, distances[closest]))
    return np.array(matches)


def ransac(landscape_1_keypoints, landscape_2_keypoints, matches, iterations=500,
    best_inliers = 0
    best_homography = None
```

```
    for _ in range(iterations):
        random_indices = np.random.choice(len(matches), 4)
        src_points = []
        dest_points = []
        for i in random_indices:
            src_points.append(landscape_1_keypoints[int(matches[i][0])].pt)
            dest_points.append(landscape_2_keypoints[int(matches[i][1])].pt)

        src_points = np.float32(src_points)
        dest_points = np.float32(dest_points)

        homography_matrix = homography_transform(src_points, dest_points)

        inliers = 0
        for i, j, _ in matches:
            src_point = landscape_1_keypoints[int(i)].pt
            dest_point = landscape_2_keypoints[int(j)].pt

            src_point = np.float32([src_point[0], src_point[1], 1])
            dest_point = np.float32([dest_point[0], dest_point[1], 1])

            transformed_point = homography_matrix @ src_point
            transformed_point /= transformed_point[2]

            dist = np.linalg.norm(transformed_point[:2] - dest_point[:2])
            if dist < threshold:
                inliers += 1

        if inliers > best_inliers:
            best_inliers = inliers
            best_homography = homography_matrix
    return best_homography


def stitch_images(image1, image2, homography_matrix):
    height, width, _ = image1.shape
    img2_warped = cv2.warpPerspective(image2, homography_matrix, (width, height))
    return np.hstack((image1, img2_warped))


landscape_1 = plt.imread('Assignment4/landscape_1.jpg')
landscape_1_keypoints, landscape_1_descriptors = extract_feature(landscape_1)

landscape_2 = plt.imread('Assignment4/landscape_2.jpg')
landscape_2_keypoints, landscape_2_descriptors = extract_feature(landscape_2)
```

```
matches = match_features(landscape_1_descriptors, landscape_2_descriptors)

homography_matrix = ransac(landscape_1_keypoints, landscape_2_keypoints, matches)
panorama = stitch_images(landscape_1, landscape_2, homography_matrix)

plt.imshow(panorama)
plt.show()
```



3(a) given that

focal point $= 721.5\,mm$
Principal point $= (609.6, 172.9)$

$$k = \begin{bmatrix} f & 0 & P_x \\ 0 & f & P_y \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow k = \begin{bmatrix} 721.5 & 0 & 609.6 \\ 0 & 721.5 & 172.9 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Since the camera's image plne is orthogonal to ground, ground plane is xz plane, with $y=0$.
i.e. equation for ground plane is $ax + 0y + cz = 0$
$\therefore$ the orthogonal vector to ground plane will be $\begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix}$
And given that camera is $1.7m$ above the ground, the magnitude for variable y in this vector is 1.7
So the equation of ground plane relative to camera is:

$$0x + 1.7y + 0z = 0$$

$$\boxed{\therefore y = -1.7}$$

(c) First we convert our principal point to homogenous coordinate, h

$$h = w\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Now using $\begin{bmatrix} wP_x \\ wP_y \\ w \end{bmatrix} = K\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \therefore \begin{bmatrix} x \\ y \\ z \end{bmatrix} = wK^{-1}\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$

converted focal length to m to keep units consistent with the camera height

$$\text{Since} \quad k = \begin{bmatrix} 0.7215 & 0 & 609.6 \\ 0 & 0.7215 & 172.9 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow K^{-1} = \begin{bmatrix} 1.386 & 0 & -0.845 \\ 0 & 1.386 & -0.24 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = w\begin{bmatrix} 1.386 & 0 & -0.845 \\ 0 & 1.386 & -0.24 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Now we know that $y = -1.7$, we can solve for w by:

$$w\left[0 \cdot P_x + 1.386 \cdot P_y + -0.24 \cdot 1\right] = -1.7$$

$$w = \frac{-1.7}{1.386P_y - 0.24}$$

Now to solve for X and Z, plug in the principal points P_x and P_y.

4 Similar triangle formula $\frac{T}{Z} = \frac{T + x_l - x_r}{Z - f}$

where x_l = x coordinate of a pixel in image from left camera
x_r = x coordinate of a pixel in image from right camera
f = focal length
T = baseline (i.e. dist b/w optical centres of the two camera's)
Z = depth

Disparity $= x_l - x_r$

Since the camera's are parallel $y_l = y_r$, we only look for corresponding x's along a horizontal line called scan line.

For each pixel (x_l, y_l) in left image along the scan line we find the corresponding best match pixel (x_r, y_r) in right image by taking a normalized correlation of the region around the desired pixel.

to calculate normalized correlation.

$$NC(region_l, region_r) = \frac{\sum_x \sum_y \left(I_{region_l}(x,y) \cdot I_{region_r}(x,y) \right)}{\| I_{region\,l} \| \cdot \| I_{region\,r} \|}$$

A best match is the patch around pixel (x_r, y_r) in right image which resulted in highest value for NC.

Complexity of NC per pixel = region_height x region_width

We do this for every pixel in the image

$\therefore$ Complexity $= O(\text{Image\_height} \times \text{Image\_width} \times \text{region\_height} \times \text{region\_width})$

Using the similar triangle formula, we can derive the formula for depth (Z):

$$Z = \frac{f \cdot T}{\underbrace{x_l - x_r}}$$ ← disparity, which we just calculated