CSC420 Assignment 4

1. Estimate the width and height of the door (in cm) from the picture.

Note: There is a manual taped on the door, with a length of 21.6cm (~816 pixels) and a width of 9.2cm (~348 pixels).

Output:

Implementation

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import transform
from scipy.spatial.distance import euclidean

if __name__ == '__main__':
    door_img = cv2.imread('door.JPG', 1)[:, :, ::-1]

    door_coordinates = np.array([[312, 473], [1681, 348],
                                 [1468, 4008], [372, 3598]])
    manual_coordinates = np.array([[1172, 1880], [1314, 1887],
                                   [1302, 2273], [1159, 2246]])

    x_manual = list(manual_coordinates[:, [0]].flatten())
    y_manual = list(manual_coordinates[:, [1]].flatten())
    x_door = list(door_coordinates[:, [0]].flatten())
    y_door = list(door_coordinates[:, [1]].flatten())

    # outline the door by green lines
    plt.plot(x_door + x_door[:1], y_door + y_door[:1], 'g')
    # outline the manual by red lines
    plt.plot(x_manual + x_manual[:1], y_manual + y_manual[:1], 'r')

    plt.imshow(door_img)
    plt.show()

    f = 38  # 1cm ~ 38 pixels
    x_real_manual = [0 + 1, 9.2 * f + 1, 9.2 * f + 1, 0 + 1]
    y_real_manual = [0 + 1, 0 + 1, 21.6 * f + 1, 21.6 * f + 1]

    src = np.vstack((x_manual, y_manual)).T
    dst = np.vstack((x_real_manual, y_real_manual)).T

    # compute homography H
    tform = transform.estimate_transform('projective', src, dst)
    src_door = np.row_stack((x_door, y_door, [1] * 4))
    x_homography = tform.params.dot(src_door)
    x_homography = x_homography / x_homography[2, :]

    # compute the coordinates of the corners
    upper_right = [x_homography[0, :][np.argmin(x_homography[1, :])],
                   x_homography[1, :][np.argmin(x_homography[1, :])]]

    bottom_right = [x_homography[0, :][np.argmax(x_homography[1, :])],
                    x_homography[1, :][np.argmax(x_homography[1, :])]]

    upper_left = [x_homography[0, :][0], x_homography[1, :][0]]

    # compute the distance and convert to cm unit
    width = euclidean(upper_left, upper_right) / f

    height = euclidean(upper_right, bottom_right) / f

    print('width of the door {}'.format(width))
    print('height f the door {}'.format(height))
```

2. Take two photos, landscape 1 and landscape 2 and stitch them into one photograph.

a. Stitched output image:

b. Implementation

```python
from SIFT_matching import *  # Implemented in A3
import numpy as np
import random


def get_matching_keypoints(img1, img2):
    """
    Return a list of matched pair tuples (keypoint_1, keypoint_2) for
    two input images.
    """
    keypoint_1, des_1 = extract_SIFT_keypoints(img1)
    keypoint_2, des_2 = extract_SIFT_keypoints(img2)

    matched_keypoints = []

    for i in range(len(keypoint_1)):
        #  match keypoint corresponding to each keypoint
        f_j, j, thresh_ratio = calculate_correspondance(des_1[i], des_2)

        # if ratio between closet and second closet match is < 0.8, keep match
        if thresh_ratio < 0.8:
            match_tuple = (keypoint_1[i], keypoint_2[j])
            matched_keypoints.append(match_tuple)

    return matched_keypoints


def calculate_homography(matched_keypoints):
    """
    Given a list of matched pair tuples (keypoint_1, keypoint_2) for both images
    return the Homography matrix (H)
    """

    length = len(matched_keypoints)
    A = np.empty((0, 9))

    for i in range(length):
        (keypoint_i, keypoint_j) = matched_keypoints[i]
        (x_j, y_j) = keypoint_i.pt
        (x_k, y_k) = keypoint_j.pt
```

```python

        # append partial to A
        a = np.array([[x_j, y_j, 1, 0, 0, 0, -x_k*x_j, -x_k*y_j, -x_k],
                      [0, 0, 0, x_j, y_j, 1, -y_k*x_j, -y_k*y_j, -y_k]])
        A = np.append(A, a, axis=0)

    # eigenvalues and eigenvectors
    w, v = np.linalg.eig(A.T @ A)

    # homography H
    i_min = np.argmin(w)
    h = v[:, i_min]
    H = np.reshape(h, (3,3))

    return H


def transform_point(H, point):
    """
    Helper to transform points.
    """
    (x, y) = point
    q = np.array([x, y, 1])
    wp = H @ q
    p = wp/wp[2]

    return p[0], p[1]


def count_inliers(H, matched_keypoints):
    """
    Helper to compute the number of inliers.
    """
    n = 0
    for i in range(len(matched_keypoints)):
        (keypoint_i, keypoint_j) = matched_keypoints[i]
        p = np.asarray(keypoint_j.pt)
        p_hat = np.asarray(transform_point(H, keypoint_i.pt))
        d = np.linalg.norm(p - p_hat)

        if d < 3:
            n += 1

```

```python
    return n


def RANSAC_homography(matched_keypoints):
    """
    Given a list of matched keypoints, return the best solution for the
    homography (H) calculated via RANSAC.
    """

    # iterate 3000 rounds to find the correct answer with 99% prob
    # assuming 20% outliers
    optimal_H = None
    max_inliers = 0
    for i in range(3000):
        # select 3 matches at random to calculate homography
        matches = random.sample(matched_keypoints, 3)
        H = calculate_homography(matches)

        # count how many matches are inliers
        num_inliers = count_inliers(H, matched_keypoints)

        # get the homography with the most inliers
        if num_inliers > max_inliers:
            max_inliers = num_inliers
            optimal_H = H

    return optimal_H


def stitch_images(H, img1, img2):
    """
    Stitch two images together using the best  homography H.
    """
    nrow_1, ncol_1 = img1.shape[0], img1.shape[1]
    nrow_2, ncol_2 = img2.shape[0], img2.shape[1]

    # transform 4 corners of the first image
    (topleft_x, topleft_y) = transform_point(H, (0, 0))
    (topright_x, topright_y) = transform_point(H, (ncol_1-1, 0))
    (bottomright_x, bottomright_y) = transform_point(H, (ncol_1-1, nrow_1-1))
    (bottomleft_x, bottomleft_y) = transform_point(H, (0, nrow_1-1))

    # calculate dimensions
```

```python
    # calculate dimensions
    min_y = min(topleft_y, topright_y, 0)
    max_y = max(bottomleft_y, bottomright_y, nrow_2-1)
    min_x = min(topleft_x, bottomleft_x, 0)
    max_x = max(topright_x, bottomright_x, ncol_2-1)

    # stitched iamge template
    nrows_stitched = max_y - min_y + 1
    ncols_stitched = max_x - min_x + 1
    shape = (int(nrows_stitched)+1, int(ncols_stitched)+1, 3)
    stitched_image = np.zeros(shape)

    # compute offset
    offset_x = 0 - min_x
    offset_y = 0 - min_y

    # transform pixels in the first image
    for x in range(ncol_1):
        for y in range(nrow_1):
            (x_t, y_t) = transform_point(H, (x,y))
            x_t = int(round(x_t + offset_x))
            y_t = int(round(y_t + offset_y))
            stitched_image[y_t, x_t, :] = img1[y, x, :]

    # shift pixels in the second image
    for x in range(ncol_2):
        for y in range(nrow_2):
            x_t = int(round(x + offset_x))
            y_t = int(round(y + offset_y))
            stitched_image[y_t, x_t, :] = img2[y, x, :]

    # interpolate skipped pixels
    for x in range(1, stitched_image.shape[1]-1):
        for y in range(1, stitched_image.shape[0]-1):
            # interpolate based on neighbours if skipped
            if np.all(stitched_image[y, x, :] == 0):
                interpolate_x = 0.5*(stitched_image[y, x-1, :] +
                                     stitched_image[y, x+1, :])
                interpolate_y = 0.5*(stitched_image[y-1, x, :] +
                                     stitched_image[y+1, x, :])
                if np.linalg.norm(interpolate_x) > np.linalg.norm(interpolate_y)
                    stitched_image[y, x, :] = interpolate_x
                else:
                    stitched_image[y, x, :] = interpolate_y

    return stitched_image


if __name__ == '__main__':
    landscape1_path = 'landscape_1.jpg'
    landscape2_path = 'landscape_2.jpg'
    img1 = cv2.imread(landscape1_path)
    img2 = cv2.imread(landscape2_path)

    matched_keypoints = get_matching_keypoints(img1, img2)
    H = RANSAC_homography(matched_keypoints)
    stitch = stitch_images(H, img1, img2)

    cv2.imwrite('stitched_output.jpg', stitch)
```

3. image um_000038.png

Q3.

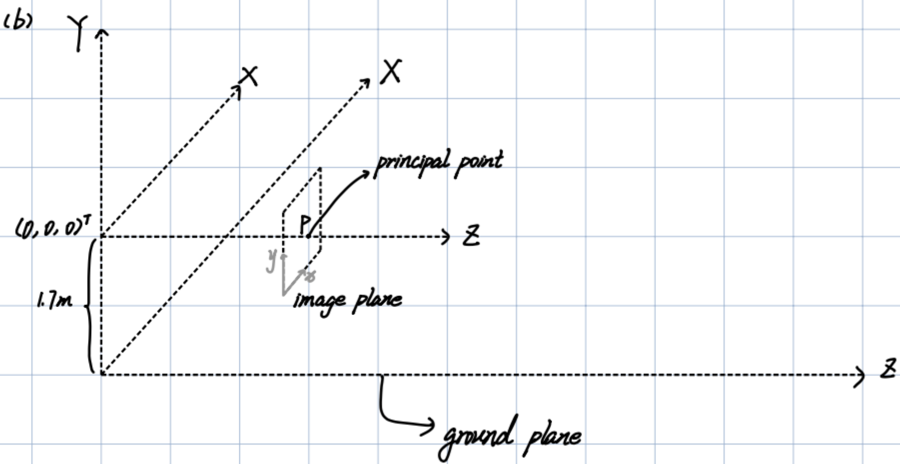
(a) focal length $f = 721.5\ mm = 0.7215\ m$

principal point $(p_x = 0.6096\ m,\ p_y = 0.1729\ m)$

$\longrightarrow$ internal camera parameter matrix

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7215 & 0 & 0.6096 \\ 0 & 0.7215 & 0.1729 \\ 0 & 0 & 1 \end{bmatrix}$$

(b)



Thus, the equation of the ground plane in camera's coordinate system is:

$$Y = -1.7$$

(c)
$$\begin{bmatrix} w\cdot x \\ w\cdot y \\ w \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \longrightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = wK^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0.7215 & 0 & 0.6096 \\ 0 & 0.7215 & 0.1729 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} wx \\ wy \\ w\cdot 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1.386 & 0 & -0.845 \\ 0 & 1.386 & -0.24 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w\cdot 1 \end{bmatrix}$$

By assuming that the point lies on the ground, $Y = -1.7 = 1.386wy - 0.24w \longrightarrow w = \frac{-1.7}{1.386y - 0.24}$

Thus, given 2D point (x, y), the 3D location would be:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.7215 & 0 & 0.6096 \\ 0 & 0.7215 & 0.1729 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} wx \\ wy \\ w\cdot 1 \end{bmatrix} = \begin{bmatrix} 1.386 & 0 & -0.845 \\ 0 & 1.386 & -0.24 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w\cdot 1 \end{bmatrix} ; \text{ where } w = \frac{-1.7}{1.386y - 0.24}$$

4. Describe (in mathematical form, no code) how to compute disparity for a pair of parallel stereo cameras.

- Point P in the world P (X, Y, Z)
- O_l: left camera center
- O_r: right camera center
- $P_l(x_l, y_l)$: point on the left image ($w_1 \times h_1$)
- $P_r(x_r, y_r)$: point on the right image ($w_2 \times h_2$)
- Since lines $O_l O_r$ and $P_l P_r$ are parallel, and O_l and O_r have the same y, then also P_l and P_r have the same y, that is, $y_l = y_r$.
- Similar triangles: $\frac{T}{Z} = \frac{T + x_l - x_r}{Z - f}$; depth and disparity are inversely proportional

- Compute_disparity(left_img, right_img):
  - For each point $P_l(x_l, y_l)$ in the left_img:
    - Find its corresponding point $P_r(x_r, y_r)$ in the right_img matching on the scanline $y_l = y_r$ by using SSD or normalized correlation
    - Calculate disparity $x_r - x_l$
    - Calculate depth $Z = \frac{f \times T}{x_r - x_l}$, where $T = O_r - O_l$
- Computation Complexity:
  - $SSD(path_l, patch_r) = \sum_x \sum_y \left(I_{patch_l}(x, y) - I_{patch_r}(x, y)\right)^2$
  - $NC(patch_l, patch_r) = \frac{\sum_x \sum_y \left(I_{patch_l}(x,y) \times I_{patch_r}(x,y)\right)}{\left|I_{patch_l}\right| \times \left|I_{patch_r}\right|}$
  - $O(w_1 w_2 h_1 h_2)$