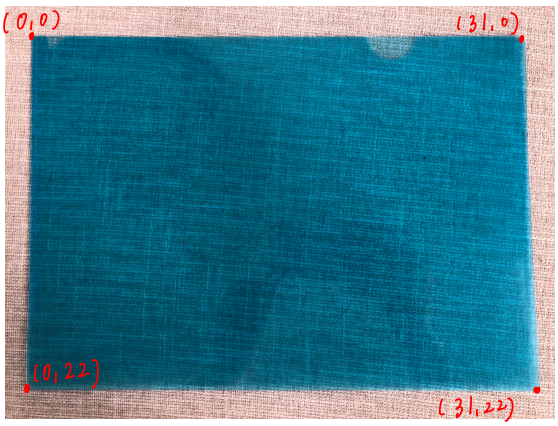
Assignment 4, Yu-Chieh Wu 10054973202

Q1. Manually labeling the corners to perform homography:

A reference of the item coordinates



*in real life, width=22, length=31

Want to perform homography transformation on item corners in the photo, to real size in life.

$$(x,y)\begin{cases}(852,519) \xrightarrow{H} (0,0)\\ (846,648) \xrightarrow{H} (0,22)\\ (999,519) \xrightarrow{H} (31,0)\\ (993,641) \xrightarrow{H} (31,22)\end{cases}(x',y')$$

plug in

plug in

Find matrix A, and H will be the eigenvector of A^TA with smallest eigenvalue

$$A=\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 1 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 1 & -x_n'x_n & -x_n'y_n & -x_1' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix}$$

Switching to python

Python code for calculating H and applying homography on door corners:

```python
def q1_a_homography():
    ps = [(852, 519), (846, 648), (999, 517), (993, 641)] # item corners in photo
    p_primes = [(0, 0), (0, 22), (31, 0), (31,22)] # itmm corners in real life
    # store door corners in homogeneous coordinate system
    door_ps = np.array([[556, 155, 1],[554, 1403, 1],[1019, 204, 1], [960, 1300, 1]])
    A = []
    for i in range(len(ps)):
        xi, yi = ps[i]
        xi_prime, yi_prime = p_primes[i]
        A += [[xi, yi, 1, 0, 0, 0, -xi_prime*xi, -xi_prime*yi, -xi_prime],\
            [0, 0, 0, xi, yi, 1, -yi_prime*xi, -yi_prime*yi, -yi_prime]]
    A = np.asarray(A)
    U, S, V = np.linalg.svd(A)
    H = V[8, :] / V[8, 8]
    H = H.reshape(3, 3)
    print('H is: \n', H, '\n')
    real_corners = []
    for corners in door_ps:
        trans = np.dot(H, corners)
        trans = trans / trans[-1]
        real_corners.append(trans)
    width = real_corners[2][0] - real_corners[0][0]
    height = real_corners[1][1] - real_corners[0][1]
    print(width, height)
    return width, height
```

H =

```
H is:
 [[ 1.63769764e-01  7.61719832e-03 -1.43485165e+02]
 [ 1.87165614e-03  1.37566726e-01 -7.29917820e+01]
 [-2.13652289e-04 -2.02459202e-05  1.00000000e+00]]
```

	door corners in photo $\xrightarrow{H}$ after homography
Upper left	(556, 155) $\xrightarrow{H}$ (-58, -57)
Lower left	(554, 1403) $\xrightarrow{H}$ (-49, 142)
Upper right	(1019, 204) $\xrightarrow{H}$ (32, -55)
Lower right	(960, 1300) $\xrightarrow{H}$ (31, 140)

After subtraction: get width of door = 90 cm, height of door = 199 cm ✓

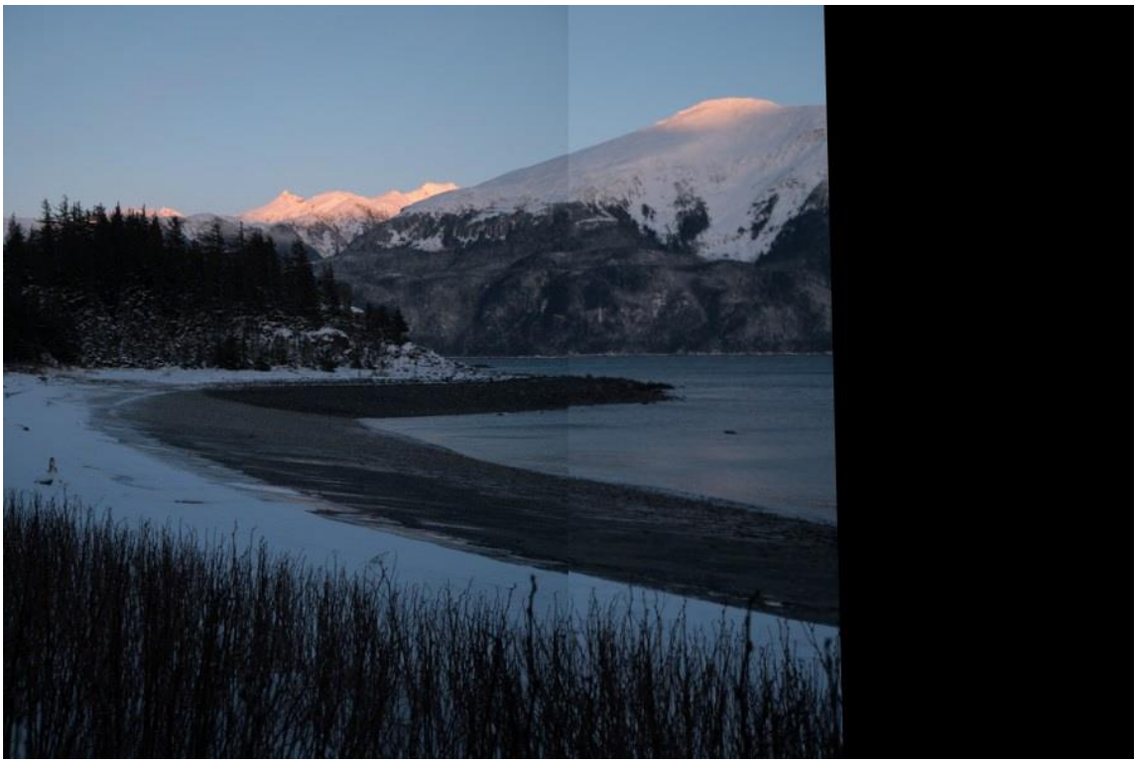
Q2. Transposing landmark2 and stitching to landmark1

```python
def q2_b_stitch_photo(img1, img2):
    all_matching = get_matching(img1, img2)
    # RANSAC to find the best homography matrix
    best_H = []
    best_inliers = 0
    for _ in range(1000):
        random_sample = sample(all_matching, 4)
        current_H = get_homography_matrix(random_sample)
        # count number of inliers
        curr_count = 0
        for j in range(len(all_matching)):
            img1_x, img1_y = all_matching[j][0].pt
            img2_x, img2_y = all_matching[j][1].pt
            img1_pt = np.array([img1_x, img1_y, 1])
            img2_pt = np.array([img2_x, img2_y, 1])
            # apply homography matrix H on img1_pt
            img1_pt = np.dot(current_H, img1_pt)
            img1_pt = img1_pt / img1_pt[-1]
            # RANSAC uses the vector norm to calculate error
            # Reference: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_\
            # and_3d_reconstruction.html#findhomography
            error = np.linalg.norm(img1_pt-img2_pt)
            if error < 5:
                curr_count += 1
        if curr_count > best_inliers:
            best_inliers = curr_count
            best_H = current_H
    # got the best homography matrix, stitch 2 imgs
    # stitching reference: https://pyimagesearch.com/2016/01/11/opencv-panorama-stitching/
    result = cv2.warpPerspective(img1, best_H,(img2.shape[1] + img1.shape[1], img1.shape[0]))
    result[0:img2.shape[0], 0:img2.shape[1]] = img2
    cv2.imwrite("./stitched.jpg", result)
```

```python
# Reference: my code for CSC420 A3, slightly modified
def get_matching(ref_img, test_img):
    sift = cv2.SIFT_create()
    ref_keypoints, ref_descriptors = sift.detectAndCompute(ref_img,None)
    test_keypoints, test_descriptors = sift.detectAndCompute(test_img,None)
    top_keypoints = []
    for i in range(ref_descriptors.shape[0]):
        # calculate Euclidean distance and fnd the ratio between closest and sec closest
        distance = np.linalg.norm(ref_descriptors[i] - test_descriptors, axis=1)
        closest_idx = np.argmin(distance)
        closest = distance[closest_idx]
        distance[closest_idx] = float('inf')
        sec_close_idx = np.argmin(distance)
        sec_close = distance[sec_close_idx]
        ratio = closest/sec_close
        if ratio > 0.8:
            continue
        else:
            top_keypoints.append([ref_keypoints[i], test_keypoints[closest_idx]])
    return top_keypoints
```

```python
# slight modification from q1
def get_homography_matrix(match_pts):
    A = []
    for i in range(len(match_pts)):
        xi, yi = match_pts[i][0].pt
        xi_prime, yi_prime = match_pts[i][1].pt
        A += [[xi, yi, 1, 0, 0, 0, -xi_prime*xi, -xi_prime*yi, -xi_prime],\
            [0, 0, 0, xi, yi, 1, -yi_prime*xi, -yi_prime*yi, -yi_prime]]
    A = np.asarray(A)
    U, S, V = np.linalg.svd(A)
    H = V[8, :] / V[8, 8]
    H = H.reshape(3, 3)
    return H
```

RESULT: Transposing landmark2 and stitching to landmark1



Q3 a) $K = \begin{bmatrix} f & 0 & P_x \\ 0 & f & P_y \\ 0 & 0 & 1 \end{bmatrix}$, $f = 721.5$, principle point = (609.6, 172.9) (P_x, P_y)

$\rightarrow K = \begin{bmatrix} 721.5 & 0 & 609.6 \\ 0 & 721.5 & 172.9 \\ 0 & 0 & 1 \end{bmatrix}$ //

b) camera coordinate system use the right hand coordinate system.
Since the camera is attached to the car at a distance of 1.7 meters above ground, the equation of the ground plane is $Y = -1.7$.

c) 3D location of any points lying on the ground has the form: $\begin{bmatrix} X \\ -1.7 \\ Z \end{bmatrix}$,
given any 2D point (x, y), we can rewrite it in homogenous system as: $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, and we know the transformation is:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = K \begin{bmatrix} X \\ -1.7 \\ Z \end{bmatrix}, \text{ therefore } \begin{bmatrix} X \\ -1.7 \\ Z \end{bmatrix} = K^{-1} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

By using python, I get $K^{-1} = \begin{bmatrix} 0.001386 & 0 & -0.844906 \\ 0 & 0.001386 & -0.239639 \\ 0 & 0 & 1 \end{bmatrix}$

$$\rightarrow \begin{bmatrix} X \\ -1.7 \\ Z \end{bmatrix} = \begin{bmatrix} 0.001386 & 0 & -0.844906 \\ 0 & 0.001386 & -0.239639 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} X \\ -1.7 \\ Z \end{bmatrix} = \begin{bmatrix} 0.001386wx - 0.844906w \\ 0.001386wy - 0.239639w \\ w \end{bmatrix}$$

Therefore: $-1.7 = 0.001386wy - 0.239639w$

$$-1.7 = (0.001386y - 0.239639)w$$

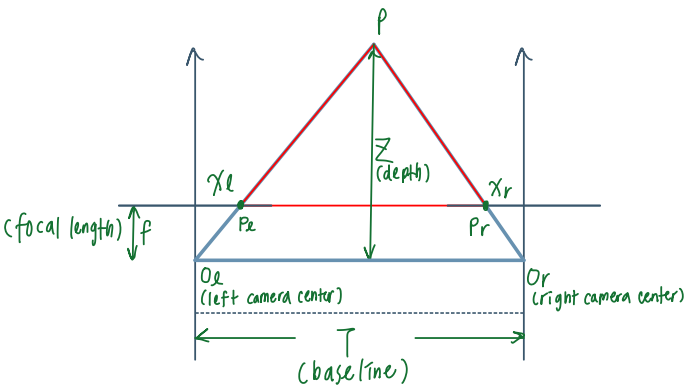
$$w = \frac{-1.7}{0.001386y - 0.239639}$$

3D point on ground: $X = 0.00138wx - 0.844906w = \dfrac{-1.7(0.00138x - 0.844906)}{0.001386y - 0.239639}$

$Z = \dfrac{-1.7}{0.001386y - 0.239639}$, $Y = -1.7$

Q4. Given a pair of parallel stereo cameras, we can compute both the disparity and depth of each pixel by using similar triangles.

graph reference: lecture 12 slides, deriving disparity & depth formula



→ By similar triangles:

$$\frac{T}{Z} = \frac{T + x_r - x_l}{Z - f}$$

$$TZ - Tf = TZ + Zx_r - Zx_l$$

$$Z(x_l - x_r) = Tf$$

$$Z = \frac{Tf}{x_l - x_r}$$

Z: depth; $x_l - x_r$: disparity

Therefore, disparity $= x_l - x_r$, depth $Z = \frac{T \cdot f}{\text{disparity}}$ (they are inversely proportional)

pseudo-code:

```
'''
q4 psudo-code
Overall complexity: O(num_row*num_col*num_col)
    - 2 for loop complexity: O(num_row*num_col)
    - find best match complexity: O(num_col)
        -> for each pixel, only need to check the patch along the scanline, therefore num_col at worst case!
'''
# loop over every single pixel in left_img
for row in left_img:
    for col in left_img:
        curr_pixel = left_img[row][col]
        xl, yl = col, row
        # returns the patch along the scanline of this pixel for matching
        curr_patch = get_current_patch(curr_pixel)
        # use SSD(find minima), Normalized Corr(find maxima) to find the best match
        # on the line yr = yl(i.e. the scanline) AND on the left of xl
        xr, yr = find_best_match(curr_patch, right_img, xl, yl) # complexity: O(num_col)
        disparity = xl-xr
        depth = focal_length*baseline / disparity
```