1. (a) Matlab code for estimating the width and length (in centimeters) of the shoe.
Computes a homography transformation from the 4 corners of the 5-dollar bill in the image to its regular rectangle dimensions in cm. Uses the homography transformation to map 3 selected camera-facing corners of the shoe to real 3D coordinates, estimates height and width by extracting lengths between 3D points.

```
function out = a3q1()

im = imread('shoe.jpg');
%im = rgb2gray(shoe_img_col);

%annotate_pts of 5-dollar-bill
imshow(im);
disp('click on the four corners of the bill. Double click the last
    point');
disp('upper-left start, go clockwise');
[x,y] = getpts();
close all;

%annotate_pts of shoe
imshow(im);
disp('click at least 3 points for height&width of shoe. Double click
    the last point');
disp('start at far right corner of shoe, annote 2 more points clockwise
    ');
[shoe_x,shoe_y] = getpts();
close all;

% display the image and picked points
figure('position', [100,100,size(im,2)*0.3,size(im,1)*0.3]);
subplot('position', [0,0,1,1]);
imshow(im);
hold on;
plot([x(:); x(1)],[y(:); y(1)],'-o','linewidth',2,'color', [1,0.1,0.1],
     'Markersize',10,'markeredgecolor',[0,0,0],'markerfacecolor'
    ,[0.5,0.0,1])
plot([shoe_x(:); shoe_x(1)],[shoe_y(:); shoe_y(1)],'-o','linewidth',2,'
    color', [1,0.1,0.1], 'Markersize',10,'markeredgecolor',[0,0,0],'
    markerfacecolor',[0.5,0.0,1])

h = 15.240;
w = 6.985;

x2 = [1, w, w, 1]';
y2 = [1, 1, h, h]';

%The U and X arguments are each 4-by-2 and define the corners
%of input and output quadrilaterals.
%compute homography from homo-bill to real life bill
tform = maketform('projective',[x,y],[x2,y2]);

% warp the shoe points according to homography
[X, Y] = tformfwd(tform, shoe_x, shoe_y);
```

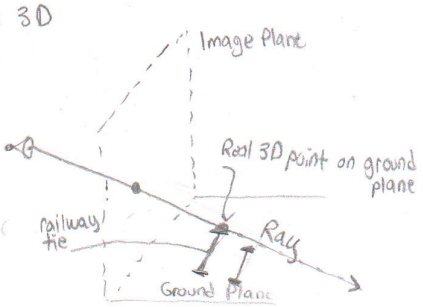
```
shoe_width = sqrt((X(3) - X(4))^2 + (Y(3) - Y(4))^2)
shoe_length = sqrt((X(2) - X(3))^2 + (Y(2) - Y(3))^2)

% Display enlarged 5 dollar bill
% h = 15240;
% w = 6985;
%
% x2 = [1, w, w, 1]';
% y2 = [1, 1, h, h]';
% [imrec] = imtransform(im, tform, 'bicubic',...
%       'xdata', [1,max(x2)],...
%       'ydata', [1,max(y2)],...
%       'size', [round(max(y2)), round(max(x2))],...
%       'fill', 0);
% figure('position', [150,150,size(imrec,2)*0.6,size(imrec,1)*0.6]);
% subplot('position', [0,0,1,1]);
% imshow(imrec)
end
```

Resulting length and width of shoe: shoe_length = 23.5997 shoe_width = 9.6821

Figure 1: Enlarged Result of homography of 5-dollar bill

2(a) Distance between adjacent railway ties

Pick 2 adjacent railway ties that are visible in the image.
We assume the rails are orthogonal to the ties, so pick the 2D intersection points of the ties with one of the rails, call them x_i, x_j. If we can calculate the 3D world coordinates for those 2D points, and find their 3D euclidean distance, this would be the distance between the adjacent ties. Say we solve for one point of $\{x_i, x_j\}$. The 3D world coordinates that map to this point are along a ray that intersects the 2D image plane at the point, which goes through the camera center and the 3D world coordinate point, but if we assume the ground plane has $Y=0$, then we can solve for the intersection between a line and the plane $Y=0$:



Given $\begin{bmatrix} x \\ y \end{bmatrix}$, K, R, t.

1. Solve for direction of Ray
2. Solve for 3D line equation of Ray
3. Find intersection between Ray and Ground Plane ($Y=0$).

$$w\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K\begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix} \rightsquigarrow \text{where } (X_R, Y_R, Z_R)^T \text{ is the 3D point in camera coordinates.}$$

$$K^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{w}\begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix}$$

the points mapping to (x,y) in 2D are on the line $P = \alpha\left(K^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right)$ in the camera coord. system, since w is unknown (we lost w when the 3D point was projected onto the 2D image plane).

1. Direction of Ray in camera coord. system:
$$(a,b,c)^T := K^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2. Equation of ray: $\alpha(a,b,c)^T = P$ (in camera coordinate system)

$$\alpha\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} R t \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

(map a point along the ray in camera coordinates to a point in world coordinates)

$$\begin{pmatrix} \alpha a \\ \alpha b \\ \alpha c \\ 1 \end{pmatrix} = Rt\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Leftrightarrow R\left(\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ 0 \end{pmatrix}\right) = \begin{pmatrix} \alpha a \\ \alpha b \\ \alpha c \\ 1 \end{pmatrix}$$

$$\alpha R^{-1}\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix} - \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ 0 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Rightarrow$$ transformation to a line with some offset (due to ray passing through camera center)

α: defines coordinate along projection ray; $R^{-1}\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$: direction of ray

To make equation more clear, convert vector $R^{-1}\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$ to cartesian coords; this is direction of ray in world coordinate system. (r_d)

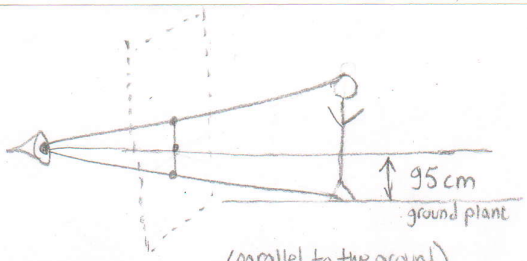
Camera center in world coordinates: $c = t^{-1}R^{-1}(0\,0\,0\,1)^T$

$\begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} = c + s(r_d)$, r_d is direction of ray, c is camera center, s is scalar parameter, $(P_x\ P_y\ P_z)^T$ defined as points that project to 2D point on image plane.

Based on the assumption that the ground plane has $Y=0$, and since 2D railway tie point lies on ground plane, we constrain $P_y = 0$.
Thus we have $0 = c_y + s R_y$ and can solve for scalar s, and solve for $(P_x\ P_y\ P_z)^T$ 3D world coordinate point.

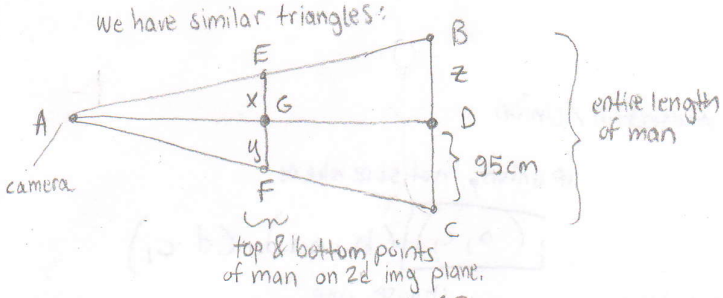
Thus we can solve for the 3D world coord points for x_i and x_j in the image, X_{i3D}, X_{j3D} and solve for their euclidean distance using square root of the norm of $(X_{i3D} - X_{j3D})$ and find the distance between adjacent railway ties.

2(b) Estimate Height of Man



- Parallel planes in 3D (parallel to the ground) have same horizon line in the image
- The image plane is orthogonal to the ground plane, thus any rays from the camera parallel to the ground plane will intersect points at 95 cm above the ground which appear along this horizon line in the image (a parallel plane to the ground intersecting the camera centre also intersects the man's torso at points in the 2D image occuring along the horizon line).

We have similar triangles:



We know lengths $EG = x$ and $GF = y$ by drawing a line from the top to the bottom of the man in the image, and selecting the top, bottom and horizon intersecting points and calculating those lengths. as they appear in the image.

By angle bisector theorem;

$$\frac{AE}{AF} = \frac{x}{y} \quad \text{and} \quad \frac{AB}{AC} = \frac{z}{95(cm)}$$

By property of similar triangles (ratio of similar sides are equal)

$$\frac{AE}{AF} = \frac{AB}{AC}$$

Thus, $\frac{x}{y} = \frac{z}{95} \Rightarrow z = \frac{x}{y} \cdot 95$

x, y are known, thus if we solve for z an estimate for the man's height is $z + 95$ (in cm)

2. PDF for this question inserted above

3. (a) Controlled test case input images and result

Figure 2: Reference image

Figure 3: Test images

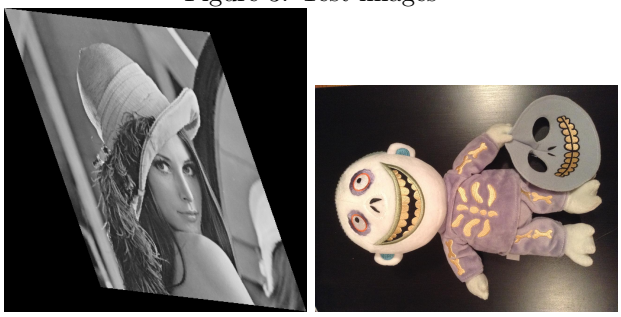


Figure 4: RANSAC algorithm result

Matlab code for RANSAC Affine Transformation Algorithm

```
function [affine_with_inliers, highest_num_inliers, mean_SSD] = ...
    ransac_affine(ref_im, test_im, num_trials, k, visualize)

if nargin<4
    k = 100; %plot keypoint transform
end;

if nargin<5
    visualize = 0; %plot keypoint transform
end;

ref_img = single(ref_im);
test_img = single(test_im);
%compute the SIFT frames (keypoints) and descriptors
[f_ref,d_ref] = vl_sift(ref_img) ;
[f_test,d_test] = vl_sift(test_img) ;

[matches, scores] = vl_ubcmatch(d_ref, d_test, 1.5);

num_matches = size(matches, 2);
matches_joined = zeros(num_matches, 3);
matches_joined(:, 1) = scores';
matches_joined(:, 2:3) = matches';
scored_matches = sortrows(matches_joined);
matches_ref = scored_matches(:, 2);
matches_test = scored_matches(:, 3);

num_matches = size(matches, 2);
%top k matches
if num_matches > k
    num_matches = k; %limit on top matches
end

sample_size = 4;
threshold = 3;
highest_num_inliers = 0; %size of largest set of inliers
affine_with_inliers = 0;
best_inlier_set = 0;
mean_SSD = Inf;

if (num_matches < sample_size) %cannot do an affine, exit
    return;
end

for i=1:num_trials %set as input to the RANSAC algo.
    sample = randsample(num_matches, sample_size);

    %compute affine transf, get transf matrix
    sr = matches_ref(sample);
    st = matches_test(sample);
    affine_vect = affine_transf(f_ref, f_test, sr, st, sample_size);
```

```
    num_inliers = 0;
    inlier_set = zeros(1, num_matches);
    for j=1:num_matches
        %The matrix f has a column for each keypoint.
        %A keypoint has center f(1:2)
        orig_point = f_ref(1:2, matches_ref(j));
        x = orig_point(1);
        y = orig_point(2);
        P_ref = [x, y, 0, 0, 1, 0; ...
                 0, 0, x, y, 0, 1;];
        Pt = P_ref*affine_vect; %compute transformed x', y'
        squared_diff = (Pt(1) - f_test(1, matches_test(j)))^2 + ...
                       (Pt(2) - f_test(2, matches_test(j)))^2;
        dist = sqrt(squared_diff);
        if (dist <= threshold)
            num_inliers = num_inliers + 1;
            inlier_set(num_inliers) = j;
        end
    end

    if highest_num_inliers < num_inliers
        highest_num_inliers = num_inliers;
        best_inlier_set = inlier_set(1:num_inliers);
    end
end

if highest_num_inliers < 3
    return
end

%refit affine to best_inlier_set (outliers removed)
affine_with_inliers = affine_transf(f_ref, f_test, ...
                                    matches_ref(best_inlier_set), ...
                                    matches_test(best_inlier_set), ...
                                    highest_num_inliers);
%calculate mean SSD with refitted affine transformation
SSD = 0;
for j=1:num_matches
    orig_point = f_ref(1:2, matches_ref(j));
    x = orig_point(1);
    y = orig_point(2);
    P_ref = [x, y, 0, 0, 1, 0; ...
        0, 0, x, y, 0, 1;];
    Pt = P_ref*affine_with_inliers; %compute transformed x', y'
    squared_diff = (Pt(1) - f_test(1, matches_test(j)))^2 + ...
                   (Pt(2) - f_test(2, matches_test(j)))^2;
    SSD = SSD + sqrt(squared_diff);
end
mean_SSD = SSD / num_matches;

end
```

Matlab code for controlled affine test case

```
function out = a3q3a(ref_im, im1, im2, num_trials, visualize)

if nargin<4
    num_trials = 5; %set a default number of trials
end;

if nargin<5
    visualize = 1; %yes, visualize the result
end;

% read images and grayscale
ref_img = rgb2gray(imread(ref_im));
img1_col = imread(im1);
affine1_img = rgb2gray(img1_col);
img2_col = imread(im2);
affine2_img = rgb2gray(img2_col);

%same number of trials for each
[affine_transf_1, ~, ssd1] = ransac_affine(ref_img, affine1_img,
   num_trials);
[affine_transf_2, ~, ssd2] = ransac_affine(ref_img, affine2_img,
   num_trials);

%minimal score is best (min mean SSD score)
if (ssd1 < ssd2)
    display_img = affine1_img;
    affine_transf = affine_transf_1;
else
    display_img = affine2_img;
    affine_transf = affine_transf_2;
end

if (visualize)
    %read the images in for visualization component
    [r, c, ~] = size(ref_img); %entire img will be mapped

    % fill in the rows of the P matrix where every 2 rows represent
    % the x,y values of one of the four corners point of ref img
    P = [1, 1, 0, 0, 1, 0; ...%top-left
        0, 0, 1, 1, 0, 1; ...
        1, r, 0, 0, 1, 0; ...%bottom-left
        0, 0, 1, r, 0, 1; ...

        c, 1, 0, 0, 1, 0; ... %top-right
        0, 0, c, 1, 0, 1; ...
        c, r, 0, 0, 1, 0; ... %bottom-right
        0, 0, c, r, 0, 1;];

    points = P*affine_transf;

    % Plot lines for affine tranformation
```

```
    imshow(display_img);
    hold on;
    line([points(1), points(3)],[points(2),points(4)],'Color','g', '
        Linewidth', 2);
    line([points(5), points(7)],[points(6),points(8)],'Color','g', '
        Linewidth', 2);
    line([points(1), points(5)],[points(2),points(6)],'Color','g', '
        Linewidth', 2);
    line([points(3), points(7)],[points(4),points(8)],'Color','g', '
        Linewidth', 2);
    hold off;

end

end
```

(b) Matlab code for assembling Shredded picture pieces

```
function out = a3q3b(num_iters, num_trials_ransac)

%downsize by half
ref_img = rgb2gray(imresize(imread('mugShot.jpg'), 0.5));

shredded_col = cell(1, 6);
shredded_col{1} = imread('cut01.png');
shredded_col{2} = imread('cut02.png');
shredded_col{3} = imread('cut03.png');
shredded_col{4} = imread('cut04.png');
shredded_col{5} = imread('cut05.png');
shredded_col{6} = imread('cut06.png');

shredded = cell(1, 6); %get grayscale
for i=1:6
    shredded{i} = imresize(shredded_col{i}, 0.5);
end

min_SSD_score = Inf;
pic = 0;

for i=1:num_iters
    sample = randsample(6, 6);
    permuted_segments = cell(1, 6);
    for j=1:6
        permuted_segments{j} = shredded{sample(j)};
    end
    permuted_img = rgb2gray(cat(2, permuted_segments{:}));
    %only get SSD score for the top 50 matched features
    [~, ~, SSD_score] = ransac_affine(ref_img, permuted_img,
        num_trials_ransac, 50);
    if min_SSD_score > SSD_score
        min_SSD_score = SSD_score;
        pic = permuted_img;
    end
end

imshow(pic);

end
```

The boundary (segments not containing the mugShot doll) in the result are still permuted because these segments don't contain any portion of the object in the reference image. Thus their order can be arbitrary since the strong relation between keypoints matches regarding the object will rebuild the object, but nothing is guaranteed for other irrelevant segments.

Figure 5: Assembled image result

4. (a) Matlab code for Helper function that computes homography transformation given 4 matches

```
% Solve for the homographic transformation between features using 4
    correspondences
function H_mat = homo_transf(f_im1, f_im2, ind1, ind2)

% We assume we are using 4 keypoint matches
k = 4;
A = zeros(2*k, 9);

for i = 1:k
    x1 = f_im1(1, ind1(i));
    y1 = f_im1(2, ind1(i));
    x2 = f_im2(1, ind2(i));
    y2 = f_im2(2, ind2(i));

    A(2*(i-1) + 1, :) = [x1 y1 1 0 0 0 -1*x2*x1 -1*x2*y1 -1*x2];
    A(2*(i-1) + 2, :) = [0 0 0 x1 y1 1 -1*y2*x1 -1*y2*y1 -1*y2];
end

%returns diagonal matrix D of eigenvalues and matrix V whose
%columns are the corresponding right eigenvectors
%[V,D] = eig(A)

M = A'*A;
[V, D] = eig(M); %eigenvalues
min_eval = Inf;
min_i = 0;
for i=1:size(V, 2) %eigenvectors
```

```
    if D(i,i) < min_eval %find min eigenvalue
        min_eval = D(i,i);
        min_i = i;
    end
end

H_vect = V(:, min_i); %eigenvector w/ min eigenvalue
H_mat = zeros(3, 3);
H_mat(1, 1:3) = H_vect(1:3);
H_mat(2, 1:3) = H_vect(4:6);
H_mat(3, 1:3) = H_vect(7:9);

end
```

Matlab code for RANSAC computed homography transformation

```
function [H_transf_best, highest_num_inliers, best_SSD] = ...
    ransac_homography(ref_im, test_im, num_trials, k, visualize)

if nargin<4
    k = Inf; %all keypoint matches
end;

if nargin<5
    visualize = 0; %plot keypoint transform
end;

ref_img = single(ref_im);
test_img = single(test_im);
%compute the SIFT frames (keypoints) and descriptors
[f_ref,d_ref] = vl_sift(ref_img) ;
[f_test,d_test] = vl_sift(test_img) ;

[matches, scores] = vl_ubcmatch(d_ref, d_test, 1.5);

num_matches = size(matches, 2);
matches_joined = zeros(num_matches, 3);
matches_joined(:, 1) = scores';
matches_joined(:, 2:3) = matches';
scored_matches = sortrows(matches_joined);
matches_ref = scored_matches(:, 2);
matches_test = scored_matches(:, 3);

%top k matches
if num_matches > k
    num_matches = k; %limit on top matches
end

threshold = 3;
highest_num_inliers = 0; %size of largest set of inliers
H_transf_best = 0;
best_SSD = Inf;
```

```
if (num_matches < 4) %cannot do a homography, exit
        return;
end

for i=1:num_trials %set as input to the RANSAC algo.
    sample = randsample(num_matches, 4);

    %compute affine transf, get transf matrix
    sr = matches_ref(sample);
    st = matches_test(sample);

    H_transf = homo_transf(f_ref, f_test, sr, st);
    num_inliers = 0;
    SSD = 0;
    for j=1:num_matches
        %The matrix f has a column for each keypoint.
        %A keypoint has center f(1:2)
        orig_point = f_ref(1:2, matches_ref(j));
        x = orig_point(1);
        y = orig_point(2);
        P_ref = [x; y; 1];
        Pt_h = H_transf * P_ref; %homogenous coords
        Pt = [Pt_h(1)/Pt_h(3), Pt_h(2)/Pt_h(3)]; %normal coords
        squared_diff = (Pt(1) - f_test(1, matches_test(j)))^2 + ...
                       (Pt(2) - f_test(2, matches_test(j)))^2;

        dist = sqrt(squared_diff);
        %dist
        if (dist <= threshold)
            num_inliers = num_inliers + 1;
        end
        SSD = SSD + sqrt(squared_diff);
    end
    mean_SSD = SSD / num_matches;
    if num_inliers > highest_num_inliers
        H_transf_best = H_transf;
        highest_num_inliers = num_inliers;
        best_SSD = mean_SSD;
    end
end

if highest_num_inliers < 3
    return
end

if (visualize)

    % Plot image
    figure;
    imshow(test_im);
    col_keypoints_test = rand(1, 3);
    col_keypoints_transf = rand(1, 3);
```

```
    for i=1:num_matches

        %Getting the transformed keypoint
        orig_point = f_ref(1:2, matches_ref(i));
        x = orig_point(1);
        y = orig_point(2);
        P_ref = [x; y; 1];
        Pt_h = H_transf_best * P_ref; %homogenous coords
        Pt = [Pt_h(1)/Pt_h(3), Pt_h(2)/Pt_h(3)]; %normal coords
        k = f_test(:, matches_test(i) );
        k(1:2) = Pt;

        h1 = vl_plotframe(f_test(:, matches_test(i) ));
        set(h1,'color',col_keypoints_test,'linewidth',6) ;
        h2 = vl_plotframe(k);
        set(h2,'color',col_keypoints_transf,'linewidth',2) ;
    end
end

end
```

Wrapper function to call main RANSAC homography function, and pass visualization flag

```
function out = a3q4a(im1, im2, num_trials)

img1 = imread(im1);
img2 = imread(im2);

ransac_homography(img1, img2, num_trials, 100, 1);
```

Figure 6: Matched SIFT keypoints

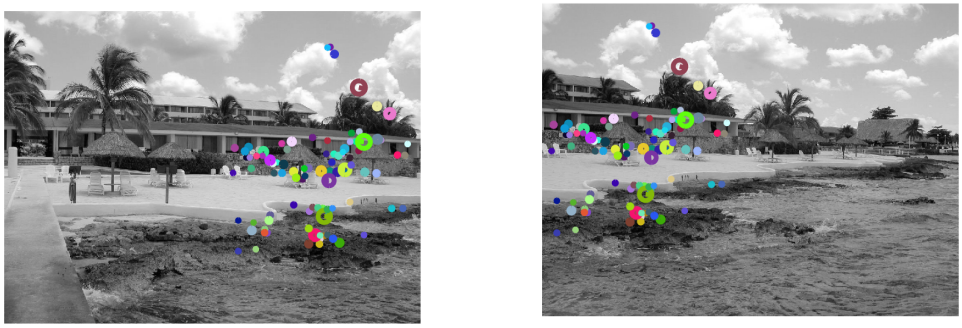


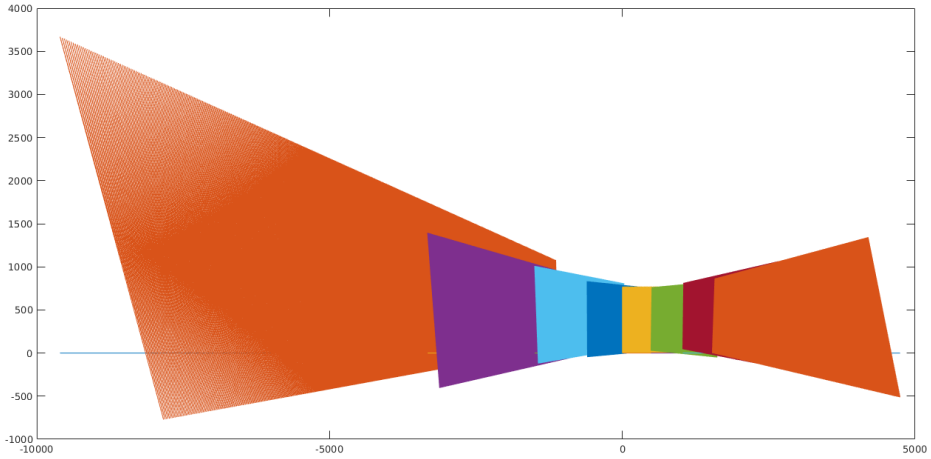
Figure 7: Visualized RANSAC homography for Right hotel image. Orginal keypoints in black, mapped keypoints via homography tranformation in yellow

(b) Matlab code to place image coordinate points onto common coordinate system

```
function out = a3q4b(num_trials)

k = 100; %limit on top matches for homography
images = cell(1, 8);

images{1} = imread('hotel-07.png');
images{2} = imread('hotel-06.png');
images{3} = imread('hotel-05.png');
images{4} = imread('hotel-04.png');
images{5} = imread('hotel-03.png'); %central image
images{6} = imread('hotel-02.png');
images{7} = imread('hotel-01.png');
images{8} = imread('hotel-00.png');

homographies = cell(1, 8);
for i=1:8
    homographies{i} = eye(3);
end

for i=1:4 % --> from left to right
    H = ransac_homography(images{i}, images{i+1}, num_trials, k);
    for j=1:i
        homographies{j} = H * homographies{j};
    end
end
for i=8:-1:6 %<-- from right to left
    H = ransac_homography(images{i}, images{i-1}, num_trials, k);
    for j=8:-1:i
        homographies{j} = H * homographies{j};
    end
end

figure
for i=1:8 %plot each img
    [x, y] = meshgrid(1:size(images{i},2), 1:size(images{i},1));
```

```
    % Reshare row and column coordinates to a flat list
    x = reshape(x,1,[]);
    y = reshape(y,1,[]);
    %homogenous untranformed coords
    untransf_img_coords(1,:) = x;
    untransf_img_coords(2,:) = y;
    untransf_img_coords(3,:) = ones(1,size(x,2));
    pan_img_homo = homographies{i} * untransf_img_coords;
    pan_x(1,:) = pan_img_homo(1,:) ./ pan_img_homo(3,:);
    pan_y(2,:) = pan_img_homo(2,:) ./ pan_img_homo(3,:);
    plot(pan_x, pan_y);
    hold on;
end
```

Figure 8: Hotel Panorama coordinate system map



(c) Matlab code to create a Panorama with sequential overlap of results

```
function out = a3q4c(num_trials)

k = 100; %limit on top matches for homography (to reduce outliers)
images = cell(1, 8);

images{1} = imread('hotel-07.png');
images{2} = imread('hotel-06.png');
images{3} = imread('hotel-05.png');
images{4} = imread('hotel-04.png');
images{5} = imread('hotel-03.png'); %central image
images{6} = imread('hotel-02.png');
images{7} = imread('hotel-01.png');
images{8} = imread('hotel-00.png');

homographies = cell(1, 8);
for i=1:8
    homographies{i} = eye(3);
end

for i=1:4 %--> from left to right
    H = ransac_homography(images{i}, images{i+1}, num_trials, k);
    for j=1:i
        homographies{j} = H * homographies{j};
    end
end
for i=8:-1:6 %<-- from right to left
    H = ransac_homography(images{i}, images{i-1}, num_trials, k);
    for j=8:-1:i
        homographies{j} = H * homographies{j};
    end
end

% Find size bounds of the panorama image in the common
% coordinate system
max_row = -Inf;
min_row = Inf;
max_col = -Inf;
min_col = Inf;

img_origins = zeros(8, 2); %row, col
for i=2:8 %each image
    unmapped_img = images{i};
    [r, c] = size(unmapped_img);

    % store the four corners of the current image
    % in homogenous coords
    % Map the corner points to the common coordinate system
    % using the corresponding homography for that image
    orig_corners = zeros(3, 4); %each column is a corner point
    orig_corners(:, 1) = [1,1,1];
    orig_corners(:, 2) = [c,1,1];
```

```
    orig_corners(:, 3) = [1,r,1];
    orig_corners(:, 4) = [c,r,1];
    corner_map = homographies{i}*orig_corners;
    corner_map(1,:) = corner_map(1,:) ./ corner_map(3,:); %space
        coordinates
    corner_map(2,:) = corner_map(2,:) ./ corner_map(3,:);


    %Find the rectangular bounds of the image to determine
    %eventual size of the panorama and the offset of the
    %individual image from the origin of the panorama area
    min_row_i = Inf;
    min_col_i = Inf;
    for j=1:4
        corner_pt = corner_map(1:2, j);
        min_row_i = floor(min(min_row_i, corner_pt(2))); %y
        min_col_i = floor(min(min_col_i, corner_pt(1))); %x

        max_row = ceil(max(max_row, corner_pt(2)));
        min_row = floor(min(min_row, corner_pt(2)));
        max_col = ceil(max(max_col, corner_pt(1)));
        min_col = floor(min(min_col, corner_pt(1)));
    end
    img_origins(i, :) = [min_row_i, min_col_i];
end

% Height and width of panorama image
panorama_height = max_row - min_row + 1;
panorama_width = max_col - min_col + 1;

eps = 10; %resize in case of slight difference with imwarp
panorama_image = zeros(panorama_height + eps, panorama_width + eps);
for i=2:8 %plot each img
    clipped_warp = imwarp(images{i}, projective2d(homographies{i}'));
    r1 = img_origins(i, 1) - min_row + 1;
    c1 = img_origins(i, 2) - min_col + 1;
    r2 = r1 + size(clipped_warp, 1) - 1;
    c2 = c1 + size(clipped_warp, 2) - 1;
    warped_image = zeros(panorama_height + eps, panorama_width + eps);
    warped_image(r1:r2, c1:c2) = double(clipped_warp);
    overlap_fix = zeros(size(warped_image));
    overlap_fix(warped_image == 0) = 1;
    panorama_image = panorama_image .* overlap_fix;
    panorama_image = panorama_image + warped_image;
end

imshow(uint8(panorama_image));
```

Figure 9: Full Hotel Panorama

Figure 10: Hotel Panorama with leftmost image removed to show the central images better