ECE 549 Computer Vision: Homework 3

Xianming Liu

1. Single View Metrology

1.1 Vanishing Points and Vanishing Lines

After determining the parallel lines in images, the vanishing points could be obtained by using cross product of two parallel lines. In my implementation, I use the criteria that the best Vanishing Points minimizes the angles between itself and center points of all the lines to choose VP. Detailed implementation is in *getVP.m*.

Following figures show the detected Vanishing points on X, Y, Z directions respectively. The vanishing points are:

X direction: $10^3 * [7.4953, 1.5383, 0.0010]$

Y direction: $10^3 * [0.0304, 1.5086, 0.0010]$

Z direction: $10^3 * [1.1216, -6.4439, 0.0010]$

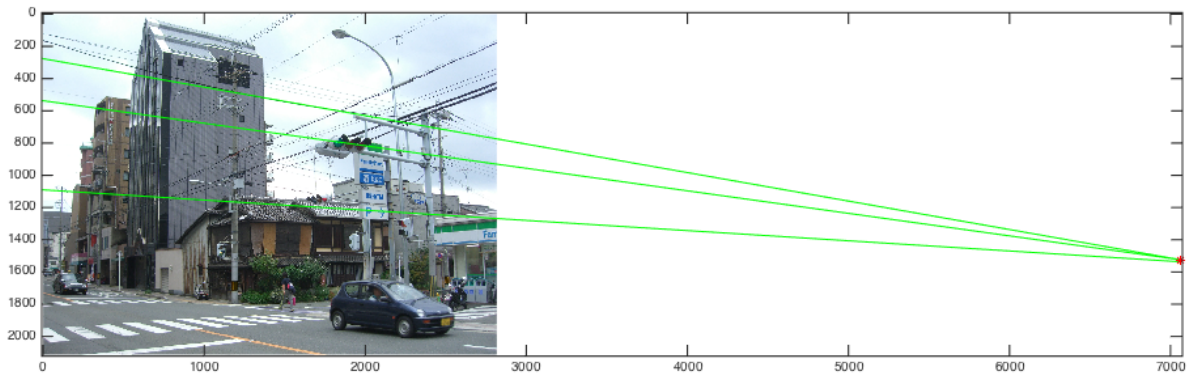


Figure 1: Vanishing Point of X direction

Figure 2: Vanishing Point of Y direction

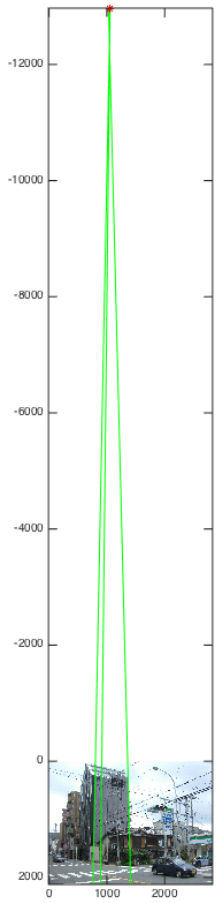


Figure 3: Vanishing Point of Z direction

Horizon vanishing line could be estimated by taking cross product of VP1 and VP2 (both in horizon directions), which is $10^3 * [1.1216, -6.4439, 0.0010]$.

Figure 4: Horizon line estimated using cross product.

1.2 Focal Length and Optical Center

Given the intrinsic matrix as

$$K = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix},$$

and vanishing points $X_i^T X_j = 0, \forall i \neq j$, we can use the estimated three vanishing points to estimate the intrinsic matrix K, by taking:

$$X_i^T X_j = p_i^T K^{-T} K^{-1} p_j = 0, \forall i \neq j,$$

because the rotation matrix $R^T R = I$. Moreover,

$$K^{-T}K^{-1} = \begin{pmatrix} 1/f^2 & 0 & -u_0/f^2 \\ 0 & 1/f^2 & -v_0/f^2 \\ -u_0/f^2 & -v_0/f^2 & \frac{u_0^2+v_0^2}{f^2}+1 \end{pmatrix}.$$

By involving symbol variables in Matlab command *solve*, we can get the solution:
$f = 2248.48$, $(u_0, v_0) = (803.52, 1247.62)$.

1.3 Rotation Matrix

Solving the rotation matrix also relies on the detected three vanishing points. Since

$$\omega \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = KR \begin{pmatrix} X \\ Y \\ Z \end{pmatrix},$$

we use the correspondences between p_1 and $[1,0,0]$, p_2 and $[0,1,0]$, p_3 and $[0,0,1]$ to solve each column of rotation matrix R, as:

$$\omega_1 * p_1 = KR \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = Kr_1$$

$$\omega_2 * p_2 = KR \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = Kr_2$$

$$\omega_3 * p_3 = KR \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = Kr_3$$

and the constraints that $R * R^T = I$ to solve rotation matrix R, and got:

$$R = \begin{pmatrix} 0.9499 & -0.3123 & -0.0119 \\ 0.0499 & 0.1140 & 0.9922 \\ 0.3085 & 0.9431 & -0.1238 \end{pmatrix},$$

1.4 Height Estimation

To estimate the height, I did the solution following these steps:

1) Estimate the horizon line:

$p_1 = 1.0e+03 * [6.3995, 1.3115, 0.0010]$,

$p_2 = 1.0e+03 * [-1.4942, 1.2491, 0.0010]$.

And horizon line is got by cross product of p_1 and p_2: $[0, -0.0008, 1.000]$.

These results are shown in following figures

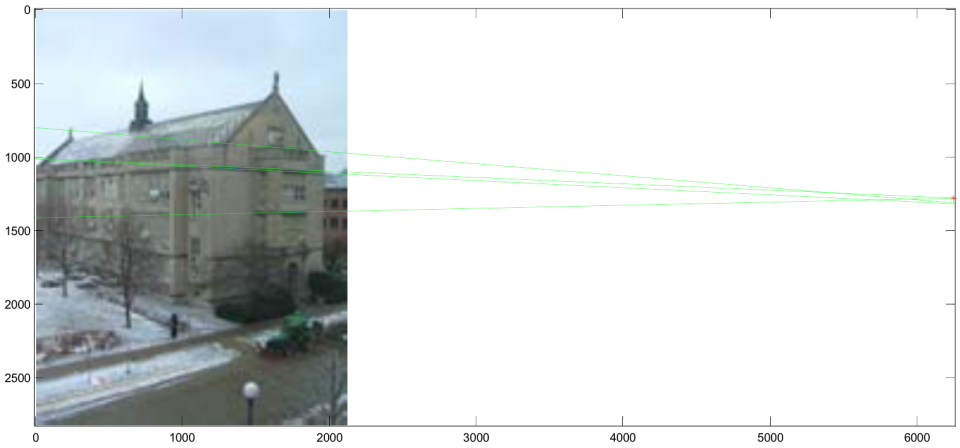


Figure 5: horizon vanishing point 1

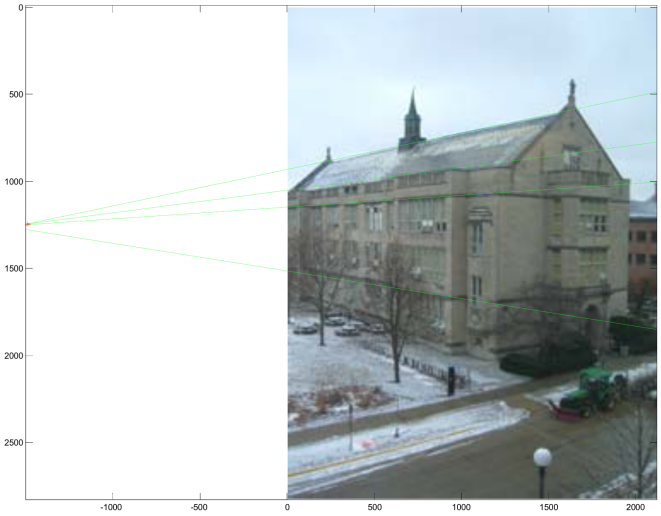


Figure 6: Horizon vanishing point 2

Figure 7: Estimated horizon line

2) Estimate the center of camera

The height of the camera is equal to the vertical distance between the horizon line and the ground. In order to estimate the height of camera, a vertical line is drawn to cross the sign and horizon line respectively, as shown in Figure 8. The Y index are used to estimate the height of the camera, $h_{camera} = (2194 - 1260)/(2194 - 2065) * h_{sign} = 11.95m$

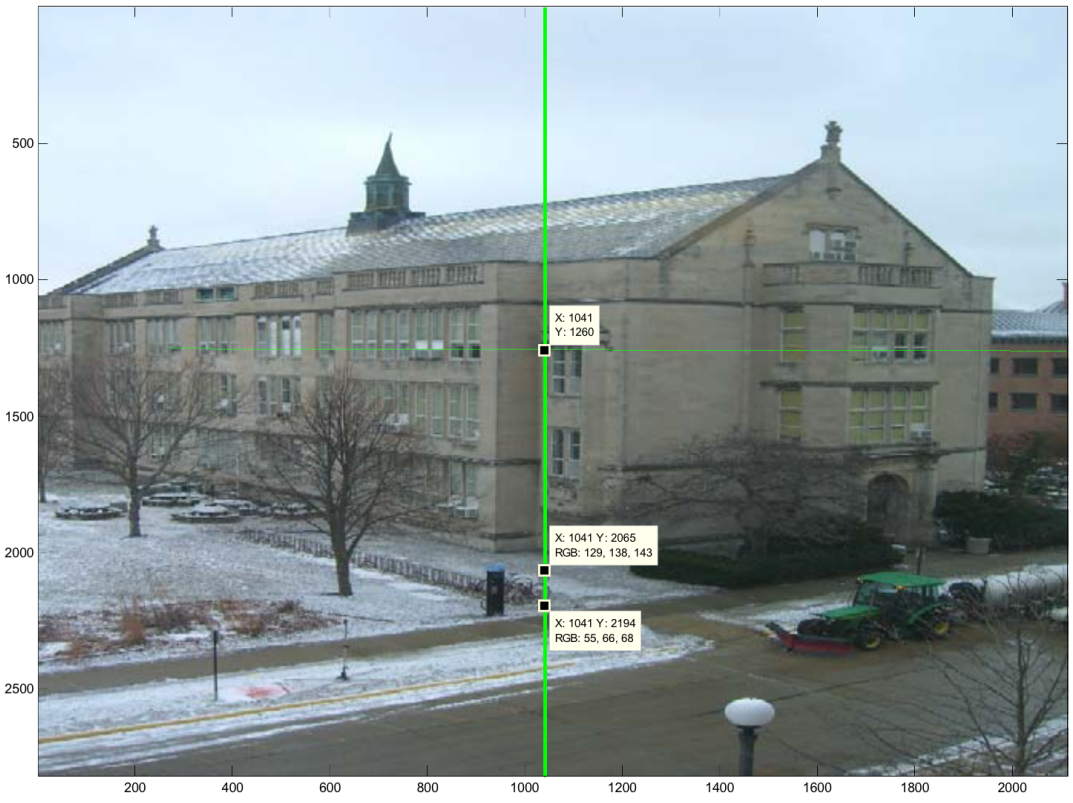


Figure 8: Estimating the height of camera

3) Estimate the height of building

Similarly, the same approach as 2) is used to estimate the height of building. A vertical line of the building is (p_0, p_1), and connect the bottom of the sign and the bottom of building $(\mathrm{p_0})$, which cross the horizon line estimated in 1) at point *O*; connect the top of the sign and point *O*, and cross vertical building line at point p_2. Then the height of building could be estimated as:

$$h_{building} = \frac{y_0 - y_1}{y_2 - y_1} * h_{sign} = 17.48m$$

Figure 9: Estimating the height of building

4) Estimate the height of truck

Use the same approach, we can also estimate the height of truck:

$$h_{truck} = (2332 - 2123)/(2332 - 2184) * h_{sign} = 2.33m$$

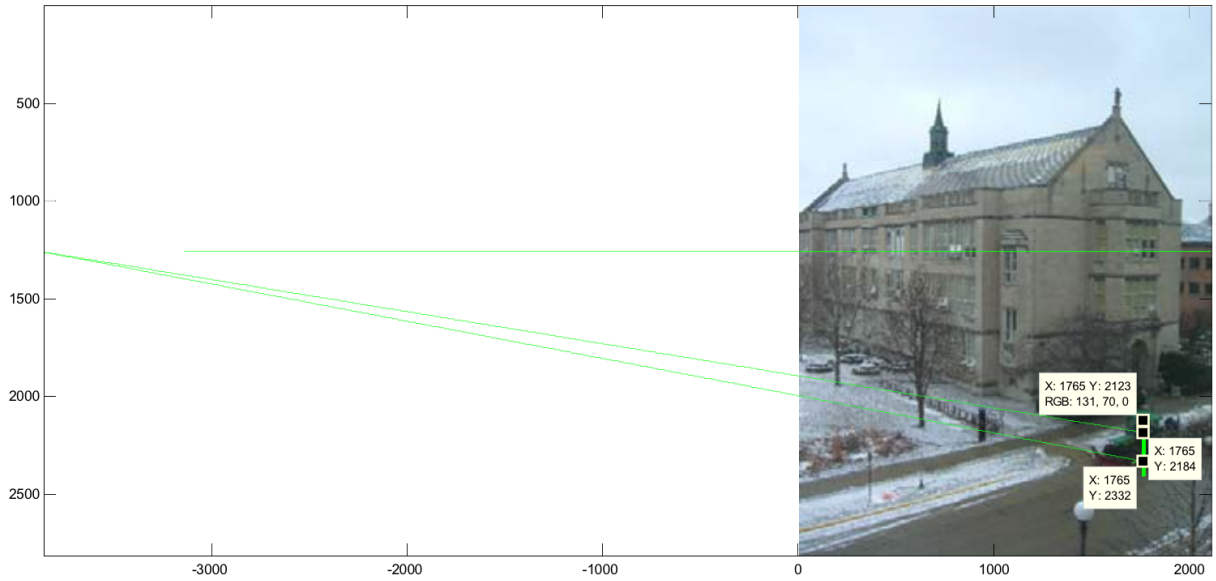


Figure 10: Estimating the height of the truck.

1.5 Extra Credit

I implemented the extra credit "camera calibration from the three detected vanishing point", in camCalibFromVP.m.

```
function [ f, u0, v0, R] = camera_calib( vp )
%CAMERA_CALIB Camera Calibration
% Homework 3, problem 1, Extra Credit E(2)
% Input:
%   vp: vanishing point, in x, y, z direction respectively, in col 1, 2,
3
% Output:
%   K - Intrinsic Matrix, R - Rotation Matrix

%% Start of function
%--------------------------------------------------%
% Solving camera focal length and optical center (u0, v0)
% Using matlab solving function solve() here
%--------------------------------------------------%
syms f u v;
% Intrinsic matrix [f, 0, u; 0, f, v; 0, 0, 1]
% using vp_i' * (K^-1)' * K^-1 * vp_j = 0
% calculate KK = (K^-1)' * K^-1
KK = [f^-2, 0, -u/f^2; 0, f^-2, -v/f^2; -u/f^2, -v/f^2, (u/f)^2+(v/
f)^2+1];
eqn1 = vp(:,1)'* KK * vp(:,2) == 0;
eqn2 = vp(:,1)'* KK * vp(:,3) == 0;
eqn3 = vp(:,2)'* KK * vp(:,3) == 0;
disp('Solving equations...');
solutions = solve(eqn1, eqn2, eqn3, f, u, v);
f = eval(vpa(solutions.f(1))); f = abs(f);
u0 = eval(vpa(solutions.u(1)));
v0 = eval(vpa(solutions.v(1)));

%--------------------------------------------------%
% Solving rotation matrix
%--------------------------------------------------%
R = zeros(3,3);
K = [f, 0, u0; 0, f, v0; 0, 0, 1];
R(:,1) = K \ vp(:,1);
R(:,2) = K \ vp(:,2);
R(:,3) = - K \ vp(:,3);
% Normalization since R*R' = I
R(:,1) = R(:,1) / norm(R(:,1));
R(:,2) = R(:,2) / norm(R(:,2));
R(:,3) = R(:,3) / norm(R(:,3));

end
```

2. Mission Possible?

1. Answer: Yes, it is possible by displaying different views in the projection screen according to the rotation of the security camera. To achieve this, we need to know the intrinsic matrix K and extrinsic matrix of the camera, which could coordinates X, Y, Z of objects and keep updating the rotation matrix R of the camera as it rotates. Based on these, update the displayed content on the projection screen, to make it "pseudo 3D".
2. Yes, the idea is similar with question 1. All we need to know is the intrinsic and extrinsic matrix of the eyeball of the guard. But different from last question, the guard could move. So in addition to estimating the rotation matrix, also need to estimate the translation, and then update display content on the projection screen.
3. No, we cannot. Since the matrices of the camera and the guard could be different, we could not display two different scene on the projection screen. But if can add glasses on the camera and the guard, like what is done in 3D movie, it is possible.

3. Epipolar Geometry

a) Estimating the fundamental matrix F

The detected matches are plotting in Figure 11.

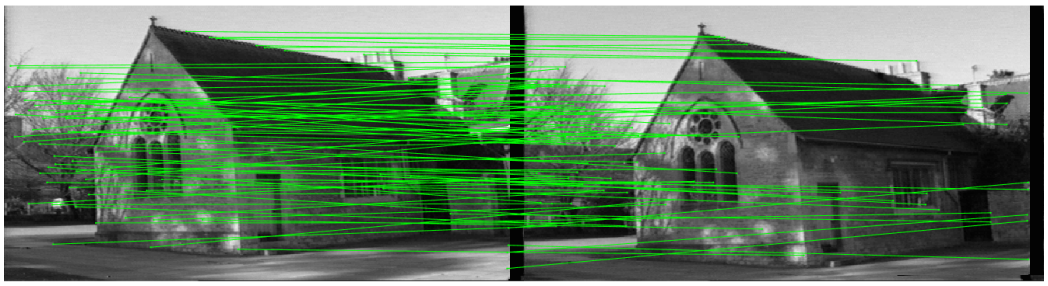


Figure 11: Matched points using RANSAC

- Indicating what test you used for deciding inlier and outlier

The criteria to decide inlier is based on estimated fundamental matrix F and the distance between matched point to epipolar line L. After estimating the matrix F, for each pair of matched points in two images, L could be calculated. If the distance that point p in image 1 to L is smaller than threshold, and the one that point p' in image 2 to L is also smaller than threshold, the pair (p, p') is an inlier.

- Display F after normalizing the unit length

$$F = \begin{pmatrix} 6.99939837293969e-07 & -8.31896213022234e-05 & 0.0211231978916213 \\ 9.07425279621264e-05 & 2.62171634979892e-06 & 0.104573375777832 \\ -0.0234184258682232 & -0.107162226550152 & -0.988223678606137 \end{pmatrix}$$

- Plot the outliers with green dots on top of the first image.

The plotted outliers are shown in Figure 12.

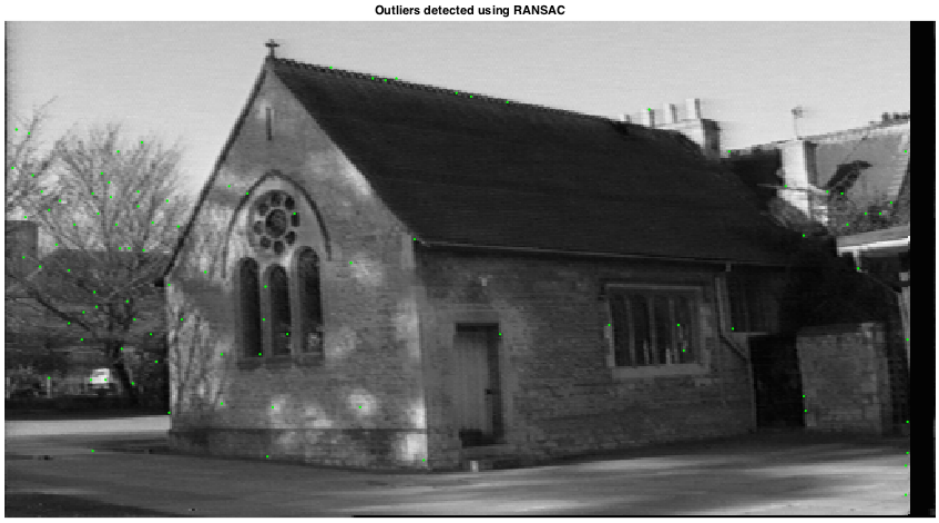


Figure 12: Detected outliers using RANSAC

b) Choose 7 sets of matching points that are well separated (can be randomly chosen).

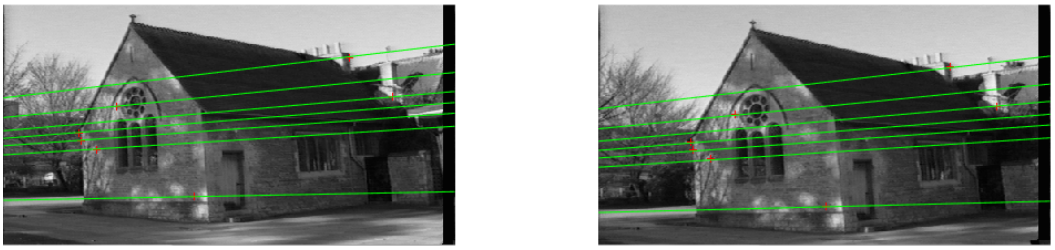
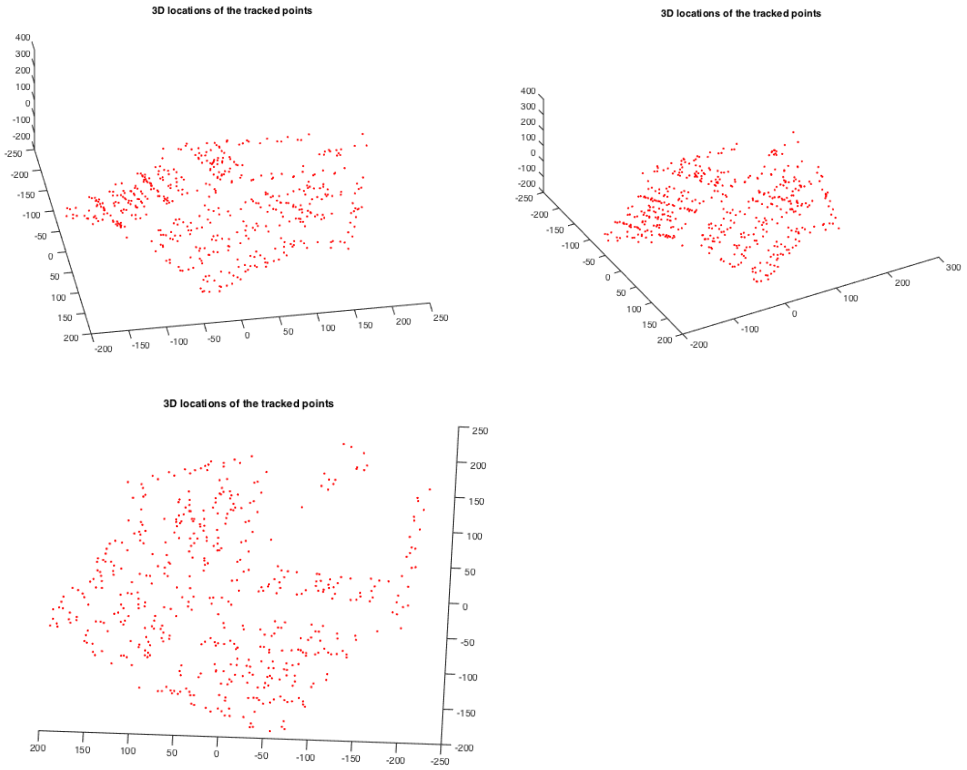


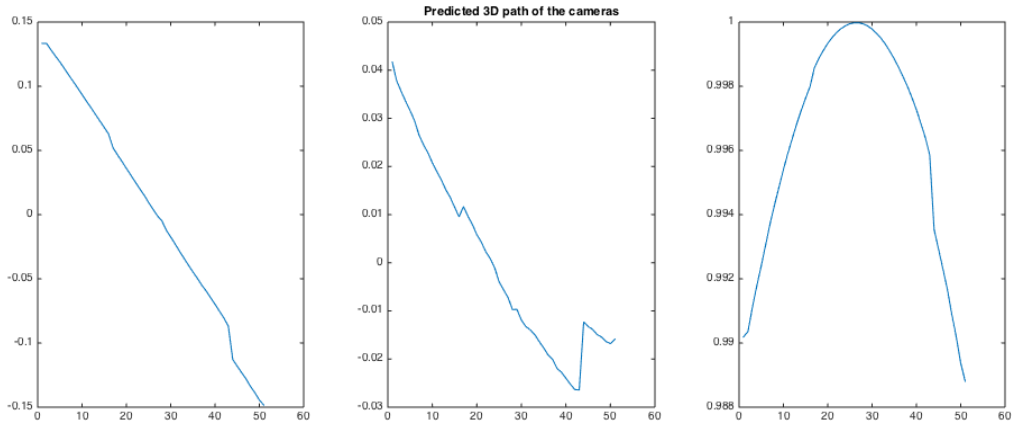
Figure 13: Epipolar lines using randomly sampled 7 matched pairs

4. Affine Structure from Motion

1) Plot the predicted 3D locations of the tracked points for 3 different viewpoints



2) Plot the predicted 3D path of the cameras.



Pseudo code:

1. For each frame: move the center of all key points to 0
2. Construct measurement matrix D, size(D) = 2m*n
3. Column j contains the projection of point j in all views
4. Row i and m+i contains coordinates of the projections of all key point in image i
5. SVD: [U W V] = svd(D);
6. U3 = U(:, 1:3), V3 = V(:,1:3), W3 = W(1:3, 1:3);
7. Affine Matrix: A = U3 * sqrt(W3); Shape matrix X = sqrt(W3) * V3';
8. Orthographic constraints on A to eliminate the affine ambiguity:
9. Solve $L = CC^T$ by least square method; (MATLAB "pinv");
10. Recover C from L: using "chol";
11. Updating:
12. A = A * C;
13. X = inv(C) * X;
14. Compute 3D Path:
15. Foreach point i:
16. ki = cross(A(i,:), A(m+i, :));
17. Normalize ki: ki = ki ./ norm(ki);
18. end
19. Return and plot