GZ720J: Homework 5
RANSAC and 3D Reconstruction

Name - Jiaxin Chen

December 1, 2016

1 Theory Questions

Question 1.1 (5pts)

Because the image coordinates are normalize and the coordinate origin (0,0) coincides with the principal point P, we can suppose:

$$p' = \begin{bmatrix} 0 \\ 0 \\ z' \end{bmatrix} \qquad p = \begin{bmatrix} 0 \\ 0 \\ z \end{bmatrix} \tag{1}$$

And the fundamental matrix satisfies the condition that for any pair of corresponding points $p \leftrightarrow p'$ in the two images:

$$p'^T F p = 0 \tag{2}$$

which is:

$$\begin{bmatrix} 0 & 0 & z' \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ z \end{bmatrix} = 0 \tag{3}$$

We can compute:

$$z' z f_{33} = 0 \tag{4}$$

Therefore, the fundamental matrix must satisfy $f_{33} = 0$.

Question 1.2 (10 pts)

Because the two cameras view an object that the second camera differs from the first by a pure translation that is parallel to the x-axis, we have:

$$R = I \qquad t = [t_x \quad 0 \quad 0]^T \tag{5}$$

The essential matrix:

$$E = [t_\times] \cdot R = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_\times \\ 0 & t_\times & 0 \end{bmatrix} \tag{6}$$

which gives us

$$E = -E^T \tag{7}$$

And for $p = [x \quad y \quad 1]^T$ and $p' = [x' \quad y' \quad 1]^T$, we also have:

$$p'^T E p = 0 \tag{8}$$

which can be wrriten as

$$[x' \quad y' \quad 1] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_\times \\ 0 & t_\times & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{9}$$

Therefore we can obtain $y = y'$, which means the epipolar lines in the two cameras are also parallel to the x-axis.

Question 1.3 (10 pts)

Because the the object and the same object viewed in a mirror satisfy, we can assume $N = [a \quad b \quad c]^T$ as the coordinate of the object, so we can have:

$$R = I - 2 \cdot N \cdot N^T = \begin{bmatrix} 1-2a^2 & -2ab & -2ac \\ -2ab & 1-2b^2 & -2bc \\ -2ac & -2bc & 1-2c^2 \end{bmatrix} \tag{10}$$

$$t = [t_x \quad t_y \quad t_z]^T = k[a \quad b \quad c]^T \tag{11}$$

Therefore, we can obtain:

$$[t]_\times = \begin{bmatrix} 0 & -kc & kb \\ kc & 0 & -ka \\ -kb & ka & 0 \end{bmatrix} \tag{12}$$

The Essential matrix:

$$E = [t]_\times \cdot R = \begin{bmatrix} 0 & -kc & kb \\ kc & 0 & -ka \\ -kb & ka & 0 \end{bmatrix} \tag{13}$$

which gives us

$$E = -E^T \tag{14}$$

Therefore, the fundamental matrix is skew-symmetric:

$$F = K^{-T}EK^{-1} = -K^{-T}EK^{-1} = -(K^{-T}EK^{-1})^T = -F^T \tag{15}$$

Question 1.4 (10 pts)

1) The epipolar line passing through the epipolar e_2 and the point p_2 can be written as:

$$l_2 = [e_2]_\times p_2 \tag{16}$$

And we have the homograph H to map p_1 to p_2:

$$p_2 = Hp_1 \tag{17}$$

Therefore the corresponding epipolar line in the second camera:

$$l_2 = [e_2]_\times Hp_1 \tag{18}$$

2) No, it's impossible to express the pipolar line similar to the case 1.

If the two cameras are seperated by a pure rotation, there is no epipolar plane or epipolar points due to the identical property of the cameras lenses' optical centers. So we cannot express the epipolar line.

2 Implementation Questions

Fundamental matrix estimation

The Eight Point Algorithm

Question 2.1 (10 pts)

After running `q2_1.m` on the pts1 and pts2 stored in `clean_correspondences.m` and `noisy_correspondenc` respectively to select points in the first image and visualize the corresponding epipolar line in the second image, we can obtain the snapshot of `displayEpipolarF.m` for $F_{8,clean}$ as Figure 1 and $F_{8,noisy}$ as Figure 2:

$$F_{8,clean} = \begin{bmatrix} -3.0021 \times 10^{-10} & -3.0692 \times 10^{-8} & 2.7052 \times 10^{-5} \\ 6.0434 \times 10^{-8} & -5.5517 \times 10^{-9} & -3.6033 \times 10^{-4} \\ -4.6576 \times 10^{-5} & 3.4853 \times 10^{-4} & -0.0054 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -0.0004 \\ 0 & 0.0003 & -0.0054 \end{bmatrix}$$

$$F_{8,noisy} = \begin{bmatrix} -1.5295 \times 10^{-8} & 6.1241 \times 10^{-9} & 1.4292 \times 10^{-6} \\ 5.7731 \times 10^{-10} & -2.4491 \times 10^{-7} & 1.7252 \times 10^{-4} \\ 2.0963 \times 10^{-5} & 1.2775 \times 10^{-4} & -0.0980 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0.0002 \\ 0 & 0.0001 & -0.0980 \end{bmatrix} \tag{19}$$

And I think the epipolar lines of `clean_correspondences.m` look more reasonable because of the very little noise in the set of correspondences. The epipolar lines of `noisy_correspondences.m` will intersect into a point, which looks much less reasonable.

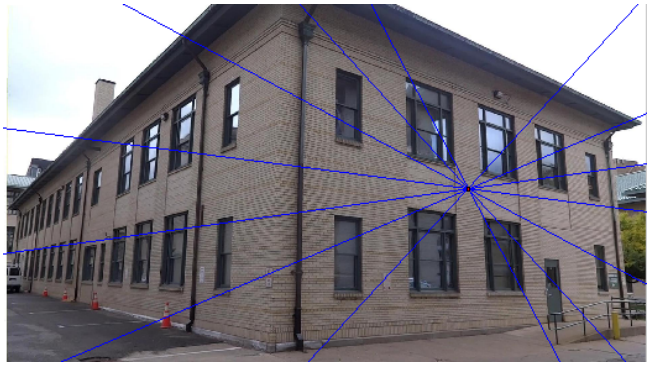
I think the $F_{8,clean}$ is more accurate. Because the accuracy of fundamental matrix F is sensitive to noise, the utilization of `clean_correspondences.m` can gurantee more accurate F than `noisy_correspondences.m`.

Blue points: selected points on first image

Blue lines: epipolar lines on second image

Figure 1: Snapshot of displayEpipolarF for $F_{8,clean}$

Blue points: selected points on first image



Blue lines: epipolar lines on second image

Figure 2: Snapshot of displayEpipolarF for $F_{8,noisy}$

```
function F = eightpoint_norm(pts1, pts2, normalization_constant)

% Compute Hartley's normalization transformation T and the corresponding
% points pst1 and pst2
T = eye(3) / normalization_constant;
T(3, 3) = 1;
pts1 = pts1 / normalization_constant;
pts2 = pts2 / normalization_constant;

% Use homogeneous linear least squares to estimate the matrix F
x1 = pts1(1, :)';
y1 = pts1(2, :)';
x2 = pts2(1, :)';
y2 = pts2(2, :)';
A = [x2.*x1 x2.*y1 x2 y2.*x1 y2.*y1 y2 x1 y1 ones(size(x1))];

[¬, ¬, V] = svd(A);
F = reshape(V(:, 9), 3, 3)';

% Compute the singular value decomposition of matrix F and set S(3, 3) = 0
[U, S, V] = svd(F);
S(3, 3) = 0;
F = U * S * V';

% Output the fundamental matrix F
F = T' * F * T;

end
```

The Seven Point Algorithm

Question 2.2 (15 pts)

After running `q2_2.m` on the pts1 and pts2 stored in `clean_correspondences.m` to select points in the first image and visualize the corresponding epipolar line in the second image, we can obtain the snapshot of `displayEpipolarF.m` for F_1 as Figure 3, F_2 as Figure 4 and F_3 as Figure 5. And from the Figure below, we can conclude that F_1 is correct.

$$
\begin{aligned}
F_1 &= \begin{bmatrix} -3.9620 \times 10^{-9} & -5.8941 \times 10^{-9} & 4.3314 \times 10^{-5} \\ 1.0013 \times 10^{-7} & -5.1.0421 \times 10^{-8} & -9.2763 \times 10^{-4} \\ -9.8376 \times 10^{-5} & 8.9909 \times 10^{-4} & -0.0203 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -0.00094 \\ -0.0001 & 0.0009 & -0.0203 \end{bmatrix} \\
F_2 &= \begin{bmatrix} -2.0300 \times 10^{-8} & 1.8840 \times 10^{-7} & -5.4223 \times 10^{-5} \\ -1.2993 \times 10^{-7} & -1.7406 \times 10^{-8} & 9.9422 \times 10^{-5} \\ 5.2403 \times 10^{-5} & -8.3739 \times 10^{-5} & -0.0070 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & -0.0001 \\ 0 & 0 & 0.0001 \\ 0.0001 & -0.0001 & -0.0070 \end{bmatrix} \\
F_3 &= \begin{bmatrix} -2.1694 \times 10^{-8} & 2.0498 \times 10^{-7} & -6.2546 \times 10^{-5} \\ -1.4957 \times 10^{-7} & -1.8002 \times 10^{-8} & 1.8707 \times 10^{-4} \\ 6.5269 \times 10^{-5} & -1.6761 \times 10^{-4} & -0.0058 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & -0.0001 \\ 0 & 0 & 0.0002 \\ 0.0001 & -0.0002 & -0.0058 \end{bmatrix}
\end{aligned} \tag{20}
$$

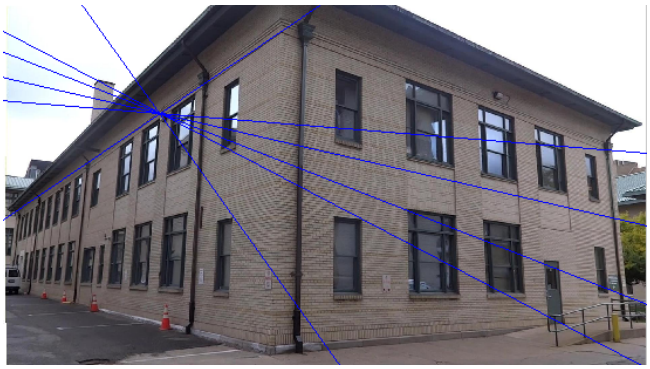
```
function F = sevenpoint_norm(pts1, pts2, normalization_constant)

% Compute Hartley's normalization transformation T and the corresponding
```

Blue points: selected points on first image

Blue lines: epipolar lines on second image

Figure 3: Snapshot of displayEpipolarF for F_1

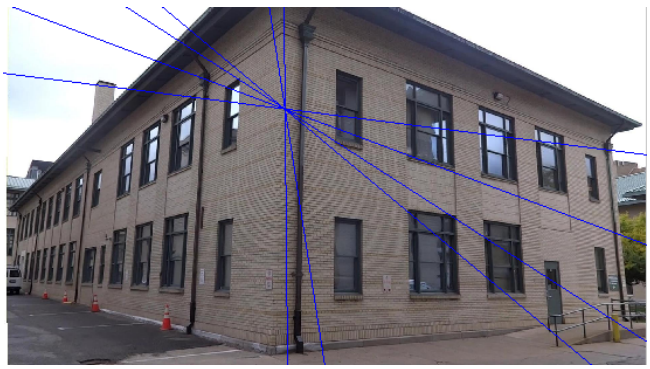
Blue points: selected points on first image



Blue lines: epipolar lines on second image

Figure 4: Snapshot of displayEpipolarF for F_2

Blue points: selected points on first image



Blue lines: epipolar lines on second image

Figure 5: Snapshot of displayEpipolarF for F_3

```
% points pst1 and pst2
T = eye(3) / normalization_constant;
T(3, 3) = 1;
pts1 = pts1 / normalization_constant;
pts2 = pts2 / normalization_constant;

% Use homogeneous linear least squares to estimate the matrix F
x1 = pts1(1, :)';
y1 = pts1(2, :)';
x2 = pts2(1, :)';
y2 = pts2(2, :)';
```

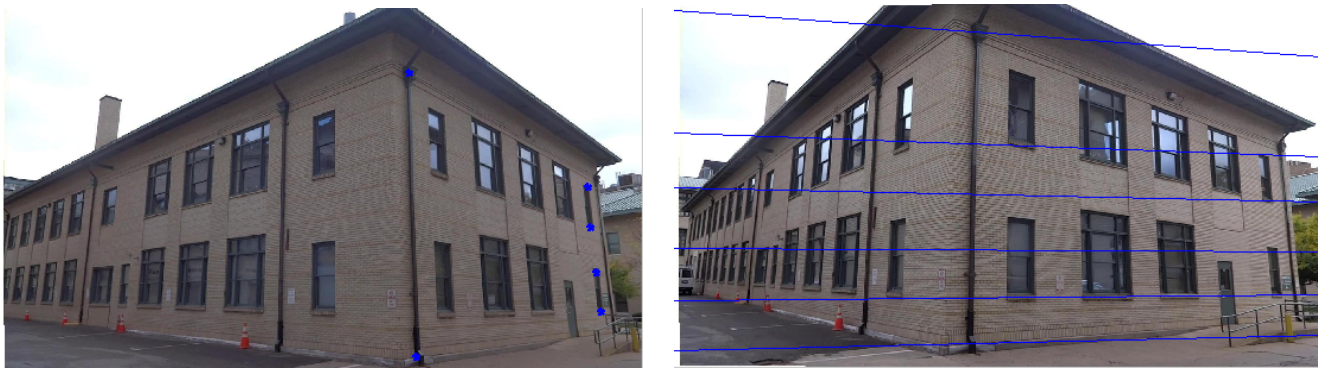
```
A = [x2.*x1 x2.*y1 x2 y2.*x1 y2.*y1 y2 x1 y1 ones(size(x1))];

[¬, ¬, V] = svd(A);
F1 = reshape(V(:, 8), 3, 3)';
F2 = reshape(V(:, 9), 3, 3)';

% Calculate lambda according to the rank-2 constraint
syms lambda;
coefficients = sym2poly(det(lambda * F1 + (1-lambda) * F2));
lambda = roots(coefficients);
lambda = lambda(lambda == real(lambda));

n = length(lambda);
F = cell(1, n);

% Output the fundamental matrix F
for i = 1 : n
    F{i} = lambda(i) * F1 + (1-lambda(i)) * F2;
    F{i} = T' * F{i} * T;
end


end
```

Computing F from Noisy Correspondences with RANSAC

Question 2.3 (10 pts)

After running `q2_3.m` on the pts1 and pts2 stored in `noisy_correspondences.m` to select points in the first image and visualize the corresponding epipolar line in the second image, we can obtain the snapshot of `displayEpipolarF.m` for $F_{7,RANSAC}$ as Figure 6. The epipolar lines look reasonable after using seven point algorithm with RANSAC. And we can obtain 41 inliers from the noisy point correspondences.

$$F_{7,RANSAC} = \begin{bmatrix} -2.2104 \times 10^{-11} & -2.8541 \times 10^{-8} & 2.3938 \times 10^{-5} \\ 5.4464 \times 10^{-8} & -5.2469 \times 10^{-9} & -3.2138 \times 10^{-4} \\ -4.2667 \times 10^{-5} & 3.1094 \times 10^{-4} & -0.0042 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -0.0003 \\ 0 & 0.0003 & -0.0042 \end{bmatrix} \tag{21}$$



Blue points: selected points on first image

Blue lines: epipolar lines on second image

Figure 6: Snapshot of displayEpipolarF for $F_{7,RANSAC}$

```
function [F, inliers] = ransacF(pts1, pts2, normalization_constant)

% Initialize parameters
iter = 10000;
threshDist = 0.001;
inliersNum = 0;
minDist = 1;
pointsNum = size(pts1, 2);

for i = 1 : iter
   % Select 7 points randomly
   index = randperm(pointsNum, 7);
   pts1Random = pts1(:, index);
   pts2Random = pts2(:, index);

   % Calculate the fundamental matrix F_Noisy
   F_Noisy = sevenpoint_norm(pts1Random, pts2Random, normalization_constant);

   for j = 1 : length(F_Noisy)
       currentInliers = [];
       totalDist = 0;
       for k = 1 : pointsNum
           % Compute the currentDist between the points with the fitting line
           p1 = [pts1(:, k); 1];
           p2 = [pts2(:, k); 1];
           currentDist = abs(p2' * F_Noisy{j} * p1);

           % Compute the inliers with currentDist smaller than threshDist
           if (currentDist < threshDist)
               currentInliers = [currentInliers, k];
           end
           totalDist = totalDist + currentDist;
       end

       % Update the inliers and minDist if better model is found
       currentNum = size(currentInliers, 2);
       if((currentNum ≥ inliersNum && totalDist < minDist) || (currentNum ...
          > inliersNum))
           F = F_Noisy{j};
           inliers = currentInliers;
           inliersNum = currentNum;
           minDist = totalDist;
           fprintf('ransacF: %i inliers, %d minDist\n', length(inliers), ...
              minDist);
       end
   end
end
end
```

Generating Novel Views of Smith Hall

Question 2.4 (30 pts)

The steps of `genNovelView.m` to create the novel views:

1) Utilize `ransacF.m` function to compute the clean fundamental matrix F and inliers from the point correspondences by choosing 7 points correspondences randomly, generating noisy F and updating inliers with minimum distance iteratively.

2) Compute the camera1 and camera2 matrix with F, K and cleaner point correspondences and implement `genM.m` to generate the camera matrix from a higher angle.

3) Generate the 3D location P of selected point correspondences.

4) Utilize `ransacPlane.m` function to choose the best plane with maximum inliers belonging to by selecting 3 points correspondences randomly, generating plane and updating inliers with minimum distance iteratively. And obtain the smith_south_plane and smith_west_plane.

5) Draw the novel view 1 of Smith Hall from camera1 viewpoint as Figure 7, the novel view 2 of Smith Hall from camera2 viewpoint as Figure 8 and the novel view 3 of Smith Hall from a higher angle as Figure 9.

Figure 7: Novel View 1 of Smith Hall from camera1

Figure 8: Novel View 2 of Smith Hall from camera2

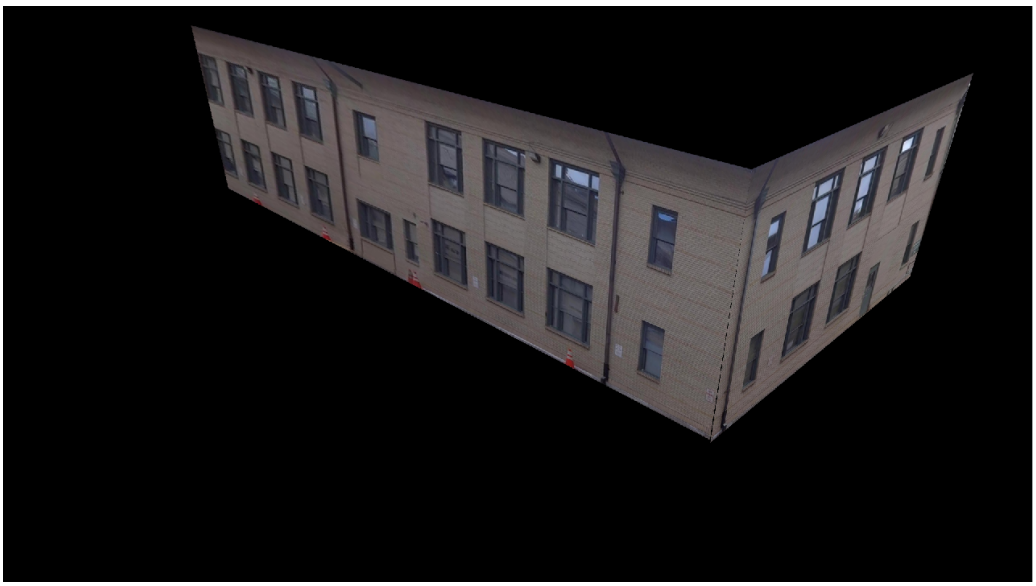


Figure 9: Novel View 3 of Smith Hall from a higher angle

```
function genNovelView

    addpath(genpath('.'));
    load('data/K.mat'); %intrinsic parameters K
    i1 = imread('data/i1.jpg');
    i2 = imread('data/i2.jpg');

    % Initialize parameters
    threshDist1 = 0.05;
    threshDist2 = 0.03;
    thetaX = 40 * pi / 180;
    thetaY = 0;
    thetaZ = 0;
    normalization_constant = max(max(size(i1), size(i2)));

    % Compute F and inliers
    load('noisy_correspondences.mat');
    [F, inliers] = ransacF(pts1, pts2, normalization_constant);

    pts1 = pts1(:, inliers);
    pts2 = pts2(:, inliers);

    % Generate the camera matrix
    M1 = K * eye(3, 4);
    M2 = camera2(F, K, K, pts1, pts2);
    M3 = genM(K, thetaX, thetaY, thetaZ);

    % Compute the 3D location P of selected point correspondences
    P = triangulate(M1, pts1, M2, pts2);

    % Generate two main planes
    [smith_south_plane, inliers1] = ransacPlane(P, threshDist1);
    P(:, inliers1) = [];
    [smith_west_plane, ¬] = ransacPlane(P, threshDist2);

    % Draw novel view 1 of Smith Hall from camera1 viewpoint
    figure(1);
    frame1 = drawNovelView(smith_south_plane, smith_west_plane, M1);
```

```
    imshow(frame1);

    % Draw novel view 2 of Smith Hall from camera2 viewpoint
    figure(2);
    frame2 = drawNovelView(smith_south_plane, smith_west_plane, M2);
    imshow(frame2);

    % Draw novel view 3 of a higher view of Smith Hall from a higher angle
    figure(3);
    frame3 = drawNovelView(smith_south_plane, smith_west_plane, M3);
    imshow(frame3);
end
```

```
function [plane, inliers] = ransacPlane(P, threshDist)

% Initialize parameters
iter = 10000;
inliersNum = 0;
minDist = 100;
pointsNum = size(P, 2);

for i = 1 : iter
   % Select 3 points randomly
   index = randperm(pointsNum, 3);
   planePoints = P(:, index)';

   % Generate plane
   normal = cross(planePoints(2, :) - planePoints(1, :), planePoints(3, ...
       :) - planePoints(1, :));
   currentPlane = [normal -dot(normal, planePoints(1, :))];
   currentPlane = currentPlane / norm(currentPlane);

   currentInliers = [];
   totalDist = 0;
   for k = 1 : pointsNum
       % Compute the currentDist between the points with the fitting line
       currentDist = abs(dot([P(:, k)', 1], currentPlane));
       currentDist = currentDist / norm(currentPlane(1:3));

       % Compute the inliers with currentDist smaller than threshDist
       if (currentDist < threshDist)
           currentInliers = [currentInliers, k];
       end
       totalDist = totalDist + currentDist;
   end

   % Update the inliers and minDist if better model is found
   currentNum = size(currentInliers, 2);
   if((currentNum ≥ inliersNum && totalDist < minDist) || (currentNum > ...
       inliersNum))
       plane = currentPlane';
       inliers = currentInliers;
       inliersNum = currentNum;
       minDist = totalDist;
       fprintf('ransacPlane: %i inliers, %d minDist\n', length(inliers), ...
           minDist);
   end
end
end
```

```
function M = genM(K, thetaX, thetaY, thetaZ)
    % Translation matrix
    t = [1; 2; 3];

    % Rotation matrix
    rotX = [1, 0, 0; 0, cos(thetaX), -sin(thetaX); 0, sin(thetaX), ...
       cos(thetaX)];
    rotY = [cos(thetaY) 0 sin(thetaY); 0 1 0; -sin(thetaY) 0 cos(thetaY)];
    rotZ = [cos(thetaZ) -sin(thetaZ) 0; sin(thetaZ) cos(thetaZ) 0; 0 0 1];
    R = rotX * rotY * rotZ;

    % Generate camera matrix
    M = K * [R, t];
end
```