CSCE 643 Multi-View Geometry CV Project IV

Fig. 1. The Checkerboard Configuration for P3P

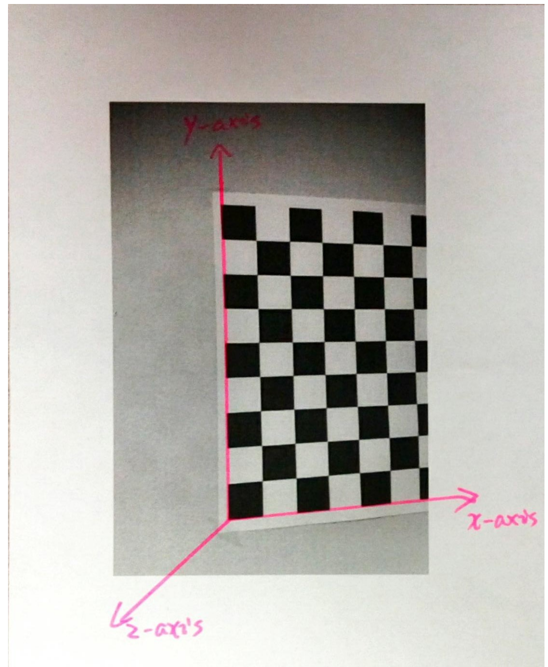

Fig. 2. The World Coordinate System for P3P

I. Checkerboard configuration. All points in 3D as a ground truth.

A. Checkerboard Picture

As the checkerboard for this time differs from last time, we retook photos using the new checkerboard and established a new coordinate system. The picture we used in this paper is shown in Figure 1.

B. World Coordinate

For establishment of world coordinate system, we choose the lower bound of checkerboard (the line formed by lower edges of black and white ceils) as the x-axis, and the leftmost counterpart as the y-axis, as shown in Figure 2. The side length of both black and white ceils is measured to be 30 mm, now we can easily build the coordinate system using mm as the unit.

C. Points Choice

Similar to previous projects, we prefer points that are not colinear for P3P processing, therefore, we selected 3 points (green marker) as A, B, C and the fourth point C marked as red, shown in Figure 3. The world coordinates (W) and coordinates in the image plane (I) are respectively shown as follows:

$$W = \begin{pmatrix} 0 & 270.0 & 0 \\ 30.0 & 60.0 & 0 \\ 180.0 & 210.0 & 0 \end{pmatrix} \tag{1}$$

$$I = \begin{pmatrix} 176.0 & 139.0 \\ 227.0 & 466.0 \\ 434.0 & 236.0 \end{pmatrix} \tag{2}$$

II. Images for P3P

As mentioned above, we adopted Figure 1 for step 2 and 3.

III. Mathematical Foundation

A. P3P System Model

We assume a pinhole camera model in this paper, moreover, the original of real-world coordinate system is assumed to be at the optical center of camera. As shown in Figure 4, when applying P3P approach, we have four maps $A \leftrightarrow u, B \leftrightarrow v, C \leftrightarrow w, D \leftrightarrow z$ that maps real-world points A, B, C, D to u, v, w, z on the image plane.

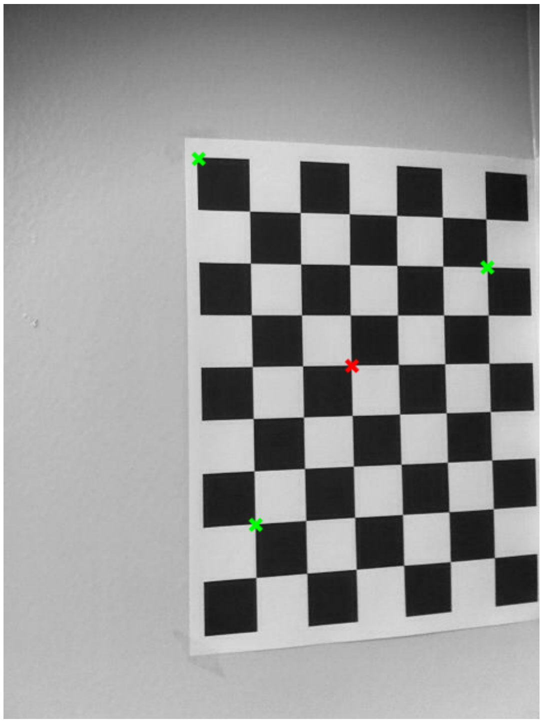

Fig. 3. Point Choice for Solving P3P

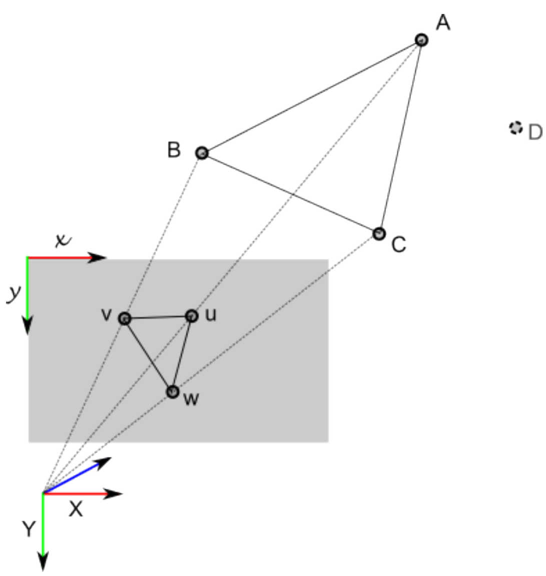

Fig. 4. 3D Real-World Points and Their 2D Counterparts on Image Plane in Camera Projection

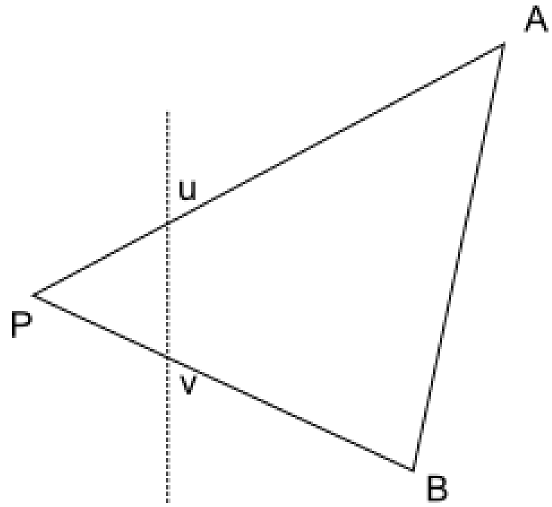

Fig. 5. Illustrating Law of Cosine in P3P Camera Model

B. Overview of Approach

In our further experiments, we will adopt three points A, B, C for solving the P3P equations and use D point and its projection for evaluation purpose. Overall, the steps we have to go through P3P are as follows:

- Acquire up to four solutions of distances $\|PA\|, \|PB\|, \|PC\|$ (P is the camera optical center).
- Convert all distance solutions mentioned above into four sets of pose configurations.
- Use the fourth point D to evaluate the choose the best pose configuration among the "up to four" solutions we have got. (Basically by applying the rotation and translation and comparing the results to real image point coordinates)

C. P3P Equation System

Though seems to be an complex approach, P3P is actually rooted into the simple law of cosine. To illustrate how we can establish the P3P equation system through law of cosine, we take $P, A \leftrightarrow u, B \leftrightarrow v$ as an example and zoom into the plane PAB, as shown in Figure 5.

As we can quickly infer accroding to geometry knowledge:

$$PA^2 + PB^2 - 2 \cdot PA \cdot PB \cdot cos\alpha_{u,v} = AB^2 \qquad (3)$$

Similarly, if we apply law of cosines to other points, we can establish the P3P equation system as follows:

$$\begin{cases} PB^2 + PC^2 - 2 \cdot PB \cdot PC \cdot cos\alpha_{v,w} - BC^2 = 0 \\ PA^2 + PC^2 - 2 \cdot PA \cdot PC \cdot cos\alpha_{u,w} - AC^2 = 0 \\ PB^2 + PA^2 - 2 \cdot PA \cdot PB \cdot cos\alpha_{u,v} - AB^2 = 0 \end{cases} \qquad (4)$$

Then, if we divide both sides of the equation system by PC^2 and suppose $v = \frac{AB^2}{PC^2}, av = \frac{BC^2}{PC^2}, bv = \frac{AC^2}{PC^2}$, we have:

$$\begin{cases} y^2 + 1 - 2 \cdot y \cdot cos\alpha_{v,w} - av = 0 \\ y^2 + 1 - 2 \cdot x \cdot cos\alpha_{u,w} - bv = 0 \\ x^2 + y^2 - 2 \cdot x \cdot y \cdot cos\alpha_{u,v} - v = 0 \end{cases} \qquad (5)$$

Apparently we can acquire $v = x^2 + y^2 - 2 \cdot x \cdot y \cdot cos\alpha_{u,v}$. By replacing v in first two equations in the system, we obtain:

$$\begin{cases} (1-a)y^2 - ax^2 - cos\alpha_{v,w}y + 2acos\alpha_{u,v}xy + 1 = 0 \\ (1-b)x^2 - by^2 - cos\alpha_{u,w}x + 2bcos\alpha_{u,v}xy + 1 = 0 \end{cases} \tag{6}$$

To acquire "up to four" solutions of lengths, we need to solve the simplified equation system mentioned above through using Wu Ritt's zero decomposition method[1].

IV. P3P Steps from Scratch

In this section, we present the detailed steps for solving P3P from scratch. The basic conditions before applying P3P is that we know 4 points (both the coordinates in the image plane and in the world plane) as mentioned in section III.

A. Normalizing the Data

The first step, similar to many other algorithms in Computer Vision, is to normalize the point coordinates, more specifically, to project image plane points u, v, w onto a unit sphere centered at camera optical center P. By applying the following equation system, we first remove unit from image points coordinates:

$$\begin{cases} u'_x = \frac{u_x - c_x}{f_x} \\ u'_y = \frac{u_y - c_y}{f_y} \\ u'_z = 1 \end{cases} \tag{7}$$

Recall that c_x, c_y are the image optical center and f_x, f_y are the focal values, both in pixels.

After removing unit, we can normalize using L2 norm as follows:

$$\begin{cases} N_u &= \sqrt{(u'_x)^2 + (u'_y)^2 + (u'_z)^2} \\ \bar{u}_x &= \frac{u'_x}{N_\mu} \\ \bar{u}_y &= \frac{u'_y}{N_\mu} \\ \bar{u}_z &= \frac{u'_z}{N_u} \end{cases} \tag{8}$$

As we will only be using normalized results $\bar{u}_x, \bar{u}_y, \bar{u}_z$ in the following paper instead of the original u_x, u_y, u_z, we replace $\bar{u}_x, \bar{u}_y, \bar{u}_z$ with u_x, u_y, u_z for simplicity.

B. The P3P Equation System

According to the simplified equation 6 we derived for P3P, we have to first compute cosine values, distances between points in the world coordinate system and a, b before continuing. The calculation of cosines can be done as show in the following equations:

$$\begin{cases} cos\alpha_{u,v} = (u_x \times v_x + u_y \times v_y + u_z \times v_z) \\ cos\alpha_{u,w} = (u_x \times w_x + u_y \times w_y + u_z \times w_z) \\ cos\alpha_{v,w} = (v_x \times w_x + v_y \times w_y + v_z \times w_z) \end{cases} \tag{9}$$

$$\begin{aligned} a_0 &= -2b + b^2 + a^2 + 1 - br^2a + 2ba - 2a \\ a_1 &= -2bqa - 2a^2q + br^2qa - 2q + 2bq + 4aq + pbr + brpa - b^2rp \\ a_2 &= q^2 + b^2r^2 - bp^2 - qpbr + b^2p^2 - br^2a + 2 - 2b^2 - abrpq + 2a^2 - 4a - 2q^2a + q^2a^2) \\ a_3 &= -b^2rp + brpa - 2a^2q + qp^2b + 2bqa + 4aq + pbr - 2bq - 2q \\ a_4 &= 1 - 2a + 2b + b^2 - bp^2 + a^2 - 2ba, \end{aligned}$$

Fig. 6. Defining a0 to a4 in P3P Quartic Polynomial

$$\begin{aligned} b_0 &= b(p^2a - p^2 + bp^2 + pqr - qarp + ar^2 - r^2 - br^2)^2, \\ b_1 &= ((1-a-b)x^2 + (qa-q)x + 1 - a + b)((a^2r^3 + 2br^3a - br^5a - 2ar^3 + r^3 + b^2r^3 - 2r^3b)x^3 + \\ &(pr^2 + pa^2r^2 - 2br^3qa + 2r^3bq - 2r^3q - 2par^2 - 2pr^2b + r^4pb + 4ar^3q + bqar^5 - 2r^3a^2q \\ &+2r^2pba + b^2r^2p - r^4pb^2)x^2 + (r^3q^2 + r^5b^2 + rp^2b^2 - 4ar^3 - 2ar^3q^2 + r^3q^2a^2 + \\ &2a^2r^3 - 2b^2r^3 - 2p^2br + 4par^2q + 2ap^2rb - 2ar^2qbp - 2p^2ar + rp^2 - br^5a + 2pr^2bq + \\ &rp^2a^2 - 2pqr^2 + 2r^3 - 2r^2pa^2q - r^4qbp)x + 4ar^3q + pr^2q^2 + 2p^3ba - 4par^2 + \\ &-2r^3bq - 2p^2qr - 2b^2r^2p + r^4pb + 2pa^2r^2 - 2r^3a^2q - 2p^3a + p^3a^2 + 2pr^2 + p^3 + 2br^3qa \\ &+2qp^2br + 4qarp^2 - 2par^2q^2 - 2p^2a^2rq + pa^2r^2q^2 - 2r^3q - 2p^3b + p^3b^2 - 2p^2brqa), \end{aligned}$$

Fig. 7. Defining b0 to b1 in P3P Quartic Polynomial

The distances we want to obtain is actually the geometric distances between three points in actual world coordinate system, thus we can easily compute them:

$$\begin{cases} \|AB\| = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2} \\ \|BC\| = \sqrt{(B_x - C_x)^2 + (B_y - C_y)^2 + (B_z - C_z)^2} \\ \|AC\| = \sqrt{(A_x - C_x)^2 + (A_y - C_y)^2 + (A_z - C_z)^2} \end{cases} \tag{10}$$

Also be noticed that the distances are not used when solving P3P, but they are necessary in computing reprojections based on P3P solutions.

Furthermore, according to the definition of a, b, we have:

$$\begin{cases} a = \frac{BC^2}{AB^2} \\ b = \frac{AC^2}{AB^2} \end{cases} \tag{11}$$

After obtaining all these data, we can now use Wu Ritt's zero decomposition method to get (x, y) solution of the simplified equation system using cosine values and a, b. One thing worth mention here is that this method might present us multi degenerated solutions to the system. However, in most of regular cases (approx. 99%), the method can yield good and realistic results. Using method from [1], we can obtain the quartic polynomial equation system:

$$\begin{cases} a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4 = 0 \\ b_0y - b_1 = 0 \end{cases} \tag{12}$$

where $\{a_0, a_1, \ldots, a_4\}$ and $\{b_0, b_1\}$ are defined in Figure 6 and 7 respectively.

To solve the quartic polynomial eqaution system, we adopted *roots* function in MATLAB, which gives us up to four solutions of x in the equation system as expected (in nondegenerated cases), using which we can further solve y and extract $\|PA\|, \|PB\|, \|PC\|$ respectively.

C. Reproject Points to 3D Space

From the last step, we should have acquired four sets of distances $\|PA\|, \|PB\|, \|PC\|$. For every set of those distances, we can easily obtain the 3D coordinates of A, B, C by multiply the $\vec{u}, \vec{v}, \vec{w}$ with corresponding distances like $\mathbf{A} = \vec{u} \cdot \|PA\|$.

D. Computing Rotation Matrix and Translation Vector

The coordinates we calculated in last section is actually in the coordinate system where camera optical center is the origin. However, the world coordinate system we build by ourself is not. Therefore, in this section we compute the rotation matrix and translation vector that can transform points from the actual camera coordinate system to the world system we established. This part can be done referring to the article in [2] [3]. The basic steps we follow is summarized below:

- Find the centroids for both coordinate systems, which can be computed simply by *mean* function in MATLAB.
- Find the optimal rotation matrix R using Singular Value Decomposition.
- Find the translation vector by $t = -R \times centroid_A + centroid_B$, where $centroid_A, centroid_B$ are centroid of point coordinates in both systems.

Iterating through all the Rotation matrix and translation vector we acquired from up to four solutions, we can compare the reprojected error of the fourth point D and simply use the least-error solution as the optimal solution of P3P.

V. Result Comparison

As we have collected all the rotation matrix and translation vectors in the above section, we can now begin to make comparisons between the results of our implemented approach and other OpenCV approaches. The methodology we used for comparing results are as follows:

- Based on the rotation matrix and translation vector we have, we can reproject the real-world point coordinates to the image space.
- After getting all the point coordinates reprojected from the last step, we can calculate the geometric distances between the fourth point and its correspondance in the image.
- The geometric distance between reprojected and actual point coordinates of the fourth point can be regarded as the error for evaluating rotation matrix and translation vector.

VI. What I Learnt from the Process

A. Data Normalization

Similar to the projects we have done before, in P3P we also did normalization for points on the image plane before we started to calculate further. This remind me the importance of data normalization. Especially when we are using different cameras, if we didn't do normalization before doing P3P, as the focal lengths and image center might differ a lot, the results we are actually getting from P3P will also vary a lot, which makes it harder for us to do further comparison between those results.

B. Law of Cosines in P3P

Though P3P seems to be difficult, but it's actually based on simple laws of cosine. The reason for such simple geometric relations is due to the underlying pinhole model, which is a basic geometric model that preserves a lot of nice properties.

C. Finding Optimal Rotation and Translation

After solving P3P, we need to calculate rotation matrix and translation vector and use them to reproject real-world points back to the image space for evaluating the errors implied in the process. In this paper, we referred to [2] as the approach for finding optimal rotation and translation between two spaces of the same dimensions, for more details about the theories in this approach please refer to [3]. This approach can be widely used in other scanarios as we have generalized it as a function to find rotation and translation relationship between any given coordinate systems.

References

[1] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[2] N. Ho, "Finding optimal rotation and translation between corresponding 3d points," http://nghiaho.com/?page_id=671.

[3] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606.

VII. Rotation and Translation Matrices

A. Results by Applying Previous Approach

By running the camera calibration approach we implemented in the previous work, we acquired the camera intrinsic matrix $\mathbf{K}$, camera projection matrix $\mathbf{P}$ and rotation matrix $\mathbf{R}$ as follows:

$$\mathbf{K} = \begin{pmatrix} 438.7795938256493 & 0 & 156.4369276062062 \\ 0 & 428.3166621327036 & 319.7357482216087 \\ 0 & 0 & 1 \end{pmatrix} \tag{13}$$

$$\mathbf{P} = \begin{pmatrix} 0.01250334096155751 & -1.508972143103616 & 0.2711271464990632 & 429.3745812434286 \\ 1.392761823410536 & -0.1564946875375584 & 1.063217835681911 & 194.2898588817989 \\ -5.201079254875831e-05 & -0.0005401809842625952 & 0.003246235734237596 & 1 \end{pmatrix} \tag{14}$$

$$\mathbf{R} = \begin{pmatrix} 0.01429199017916619 & -0.98637248668008 & -0.1639056330248407 \\ 0.9997729828680795 & 0.01150624289447384 & 0.01793290555141357 \\ -0.01580258661679046 & -0.1641247205481505 & 0.9863130103375954 \end{pmatrix} \tag{15}$$

The translation vector $\mathbf{t}$ we obtained using previous method is as follows:

$$\mathbf{t} = \begin{pmatrix} 188.9956201169497 \\ -88.98691955355272 \\ 303.8328362709736 \end{pmatrix} \tag{16}$$

B. Results from Out Implemented P3P Approach

The rotation matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{pmatrix} 0.8968 & 0.0336 & 0.4411 \\ -0.1628 & -0.9021 & 0.3997 \\ 0.4114 & -0.4303 & -0.8035 \end{pmatrix} \tag{17}$$

The rotation matrix $\mathbf{t}$:

$$\mathbf{t} = \begin{pmatrix} -41.14 \\ 161.9 \\ 285.0 \end{pmatrix} \tag{18}$$

C. Results from OpenCV P3P Approach

The rotation matrix $\mathbf{R}$:

$$\mathbf{t} = \begin{pmatrix} 0.9751724927065563 & -0.03073146620254604 & 0.2193038678489812 \\ -0.06618979781684881 & -0.9855013514892844 & 0.1562241878767766 \\ 0.2113232598022431 & -0.1668612093862306 & -0.9630679190320474 \end{pmatrix} \tag{19}$$

The rotation matrix $\mathbf{t}$:

$$\mathbf{t} = \begin{pmatrix} 19.44234760445592 \\ 160.6071508385414 \\ 295.0200953938722 \end{pmatrix} \tag{20}$$

D. Results from OpenCV Iterative Approach

The rotation matrix $\mathbf{R}$:

$$\mathbf{t} = \begin{pmatrix} 0.9877447941077802 & -0.01613246165380006 & 0.1552416355040364 \\ -0.01456187073648512 & -0.9998306912563339 & -0.01124903296001547 \\ 0.1553968263306344 & 0.008850565115871789 & -0.9878124790988907 \end{pmatrix} \tag{21}$$

The rotation matrix $\mathbf{t}$:

$$\mathbf{t} = \begin{pmatrix} 16.00623525135875 \\ 154.3321069948261 \\ 271.0935835353657 \end{pmatrix} \tag{22}$$

E. Results from OpenCV EPNP Approach

The rotation matrix $\mathbf{R}$:

$$\mathbf{t} = \begin{pmatrix} 0.9906641741585167 & -0.01403403102579241 & -0.1356006637594024 \\ -0.01047654084380424 & -0.9995828747533818 & 0.02691316762689056 \\ -0.1359218015285684 & -0.02524128508876093 & -0.9903979712198003 \end{pmatrix} \tag{23}$$

The rotation matrix $\mathbf{t}$:

$$\mathbf{t} = \begin{pmatrix} 15.3524522558304 \\ 153.5550546788758 \\ 299.6401600830907 \end{pmatrix} \tag{24}$$

Appendix

A. P3P Main Function

```
clear all; close all; clc;
img = imread('pics/p3p.jpg');
img = rgb2gray(img);
figure(1); imshow(img);
hold on;

% get all corner points in the single checker board figure
raw_corners = get_corners(img);

% corners sorted by x, 1st row
% sort by first coordinate (x) we get points by vertical lines
rawVCorners = sortrows(raw_corners, 1);

%h = plot(rawVCorners(:, 1), rawVCorners(:, 2), 'x', 'Color', 'r', 'MarkerSize', 15);
%set(h,'linewidth',3);

% count of all corner pts on the image
corners_size = size(rawVCorners);
ptsCount = corners_size(1);

vLinesCount = 8;
hLinesCount = 10;

sortedVCorners = [];
% do sorting for each vertical line
for i = 1 : vLinesCount
    sIndex = ( i - 1) * hLinesCount + 1;
    endIndex = i * hLinesCount ;
    tmpVPtsSet = rawVCorners ( sIndex : endIndex , :) ;
    tmpVPtsSet = sortrows ( tmpVPtsSet , 2) ;
    sortedVCorners = [ sortedVCorners ; tmpVPtsSet ];
end

world = [];
for i = 1 : vLinesCount
    for j = 1 : hLinesCount
       % TODO
       world = [world;
            (i - 1) * 30 (270 - 30 * (j - 1)) 0];
    end
end
% test the sequence of pts
for i = 1 : ptsCount
    h = plot(sortedVCorners(i, 1), sortedVCorners(i, 2), 'x', 'Color', 'r', '
        MarkerSize', 6);
    set(h, 'linewidth', 3);
end

%c_x = 156.4369276062062;
global c_x c_y f_x f_y;
c_x = 239.43;
c_y = 319.7357482216087;
f_x = 438.7795938256493;
```

```
f_y = 428.3166621327036;

mu0 = sortedVCorners(1, 1);
mv0 = sortedVCorners(1, 2);

mu1 = sortedVCorners(18, 1);
mv1 = sortedVCorners(18, 2);

mu2 = sortedVCorners(63, 1);
mv2 = sortedVCorners(63, 2);

imgOrig = [sortedVCorners(1, :); sortedVCorners(18, :); sortedVCorners(63, :)];

[mu0, mv0, mk0] = p3pNorm(mu0, mv0, c_x, c_y, f_x, f_y);
[mu1, mv1, mk1] = p3pNorm(mu1, mv1, c_x, c_y, f_x, f_y);
[mu2, mv2, mk2] = p3pNorm(mu2, mv2, c_x, c_y, f_x, f_y);

X0 = world(1, 1);
Y0 = world(1, 2);
Z0 = world(1, 3);

X1 = world(18, 1);
Y1 = world(18, 2);
Z1 = world(18, 3);

X2 = world(63, 1);
Y2 = world(63, 2);
Z2 = world(63, 3);

worldX = [X0, Y0, Z0; X1, Y1, Z1; X2, Y2, Z2];

distances = [];
distances = [distances; sqrt((X1 - X2)^2 + (Y1 - Y2)^2 + (Z1 - Z2)^2)];
distances = [distances; sqrt((X0 - X2)^2 + (Y0 - Y2)^2 + (Z0 - Z2)^2)];
distances = [distances; sqrt((X1 - X0)^2 + (Y1 - Y0)^2 + (Z1 - Z0)^2)];

cosines = [];
cosines = [cosines; (mu1*mu2 + mv1*mv2 + mk1*mk2)];
cosines = [cosines; (mu0*mu2 + mv0*mv2 + mk0*mk2)];
cosines = [cosines; (mu1*mu0 + mv1*mv0 + mk1*mk0)];
% solve the length of PA PB and PC
lengths = lengthSolver(distances, cosines);

imgX = [];
Rs = [];
ts = [];
% compute the Rotation and translation for each solution
for i = 1 : 4
    imgX = [lengths(i, 1)*[mu0, mv0, mk0];
        lengths(i, 2)*[mu1, mv1, mk1];
        lengths(i, 3)*[mu2, mv2, mk2]];
    [R, t] = rigid_transform_3D(worldX, imgX);
    Rs = [Rs; R];
    ts = [ts; t];
end
```

```matlab
global X3 Y3 Z3 mu3 mv3;
X3 = world(35, 1);
Y3 = world(35, 2);
Z3 = world(35, 3);
mu3 = sortedVCorners(35, 1);
mv3 = sortedVCorners(35, 2);

min_reproj = 999999999;
ns = 0;
% iterate through all four solutions to find the one that produces the
% least reprojection errors
for i = 1 : 4
    basicIndex = (i - 1) * 3;
    X3p = Rs((basicIndex + 1), 1) * X3 + Rs((basicIndex + 1), 2) * Y3 + Rs((basicIndex + 1), 3) * Z3 + ts((basicIndex + 1));
    Y3p = Rs((basicIndex + 2), 1) * X3 + Rs((basicIndex + 2), 2) * Y3 + Rs((basicIndex + 2), 3) * Z3 + ts((basicIndex + 2));
    Z3p = Rs((basicIndex + 3), 1) * X3 + Rs((basicIndex + 3), 2) * Y3 + Rs((basicIndex + 3), 3) * Z3 + ts((basicIndex + 3));

    mu3p = c_x + f_x * X3p / Z3p;
    mv3p = c_y + f_y * Y3p / Z3p;
    reproj = (mu3p - mu3) * (mu3p - mu3) + (mv3p - mv3) * (mv3p - mv3);

    if (i == 0 || abs(min_reproj) > abs(reproj))
            ns = i;
            min_reproj = reproj;
    end
end

err_mine = rep_error(Rs(4:6, :), ts(4:6));
% R from OpenCV P3P
R_p3p = [0.9751724927065563, -0.03073146620254604, 0.2193038678489812;
 -0.06618979781684881, -0.9855013514892844, 0.1562241878767766;
 0.2113232598022431, -0.1668612093862306, -0.9630679190320474];
t_p3p = [19.44234760445592, 160.6071508385414, 295.0200953938722];
err_p3p = rep_error(R_p3p, t_p3p);

R_epnp = [0.9906641741585167, -0.01403403102579241, -0.1356006637594024;
 -0.01047654084380424, -0.9995828747533818, 0.02691316762689056;
 -0.1359218015285684, -0.02524128508876093, -0.9903979712198003];
t_epnp = [15.3524522558304, 153.5550546788758, 299.6401600830907];
err_epnp = rep_error(R_epnp, t_epnp);

% calculate reprojection errors
function err = rep_error(R, t)
    global c_x f_x c_y f_y;

    global mu3 mv3 X3 Y3 Z3;
    X3p = R(1, 1) * X3 + R(1, 2) * Y3 + R(1, 3) * Z3 + t(1);
    Y3p = R(2, 1) * X3 + R(2, 2) * Y3 + R(2, 3) * Z3 + t(2);
    Z3p = R(3, 1) * X3 + R(3, 2) * Y3 + R(3, 3) * Z3 + t(3);

    mu3p = c_x + f_x * X3p / Z3p;
    mv3p = c_y + f_y * Y3p / Z3p;
    err = (mu3p - mu3) * (mu3p - mu3) + (mv3p - mv3) * (mv3p - mv3);
```

```
end

% expects row data, find rotation and translation between any two
% coordinate system based on a set of point correspondances
function [R,t] = rigid_transform_3D(A, B)
    if nargin ~= 2
            error('Missing parameters');
    end

    %assert(size(A) == size(B));

    centroid_A = mean(A);
    centroid_B = mean(B);

    N = size(A,1);

    H = (A - repmat(centroid_A, N, 1))' * (B - repmat(centroid_B, N, 1));

    [U,S,V] = svd(H);

    R = V*U';

    if det(R) < 0
        fprintf('Reflection detected\n');
        V(:,3) = V(:,3) * -1;
        R = V*U';
    end

    t = -R*centroid_A' + centroid_B';
end

function lengths = lengthSolver(distances, cosines)
    p = cosines(1) * 2;
    q = cosines(2) * 2;
    r = cosines(3) * 2;

    a = (distances(1)^2) / (distances(3)^2);
    b = (distances(2)^2) / (distances(3)^2);

    a2 = a * a, b2 = b * b, p2 = p * p, q2 = q * q, r2 = r * r;
    pr = p * r, pqr = q * pr;

    if (p2 + q2 + r2 - pqr - 1 == 0)
        error('failed to pass reality check');
    end

    ab = a * b, a_2 = 2*a;
    A = -2 * b + b2 + a2 + 1 + ab*(2 - r2) - a_2;
    if (A == 0)
        error('A is 0!');
    end

    a_4 = 4*a;
    B = q*(-2*(ab + a2 + 1 - b) + r2*ab + a_4) + pr*(b - b2 + ab);
    C = q2 + b2*(r2 + p2 - 2) - b*(p2 + pqr) - ab*(r2 + pqr) + (a2 - a_2)*(2 + q2) +
        2;
```

```
    D = pr*(ab-b2+b) + q*((p2-2)*b + 2 * (ab - a2) + a_4 - 2);
    E = 1 + 2*(b - a - ab) + b2 - b*p2 + a2;

    temp = (p2*(a-1+b) + r2*(a-1-b) + pqr - a*pqr);
    b0 = b * temp * temp;
    if (b0 == 0)
        error('b0 equals to 0!');
    end
    quartic_roots = roots([A, B, C, D, E]);
    % check if roots contain solutions
    r3 = r2*r, pr2 = p*r2, r3q = r3 * q;
    inv_b0 = 1. / b0;
    lengths = [];
    for i = 1 : 4
        x = quartic_roots(i);
        if(x <= 0)
            continue;
        end

        x2 = x*x;
        b1 = ((1-a-b)*x2 + (q*a-q)*x + 1 - a + b) * ...
            (((r3*(a2 + ab*(2 - r2) - a_2 + b2 - 2*b + 1)) * x + ...
            (r3q*(2*(b-a2) + a_4 + ab*(r2 - 2) - 2) + pr2*(1 + a2 + 2*(ab-a-b) + r2*(b - b2) + b2))) * x2 + ...
            (r3*(q2*(1-2*a+a2) + r2*(b2-ab) - a_4 + 2*(a2 - b2) + 2) + r*p2*(b2 + 2*(ab - b - a) + 1 + a2) + pr2*q*(a_4 + 2*(b - ab - a2) - 2 - r2*b)) * x + ...
            2*r3q*(a_2 - b - a2 + ab - 1) + pr2*(q2 - a_4 + 2*(a2 - b2) + r2*b + q2*(a2 - a_2) + 2) + ...
            p2*(p*(2*(ab - a - b) + a2 + b2 + 1) + 2*q*r*(b + a_2 - a2 - ab - 1)));
        if (b1 <= 0)
            continue;
        end
        y = inv_b0 * b1;
        v = x2 + y*y - x*y*r;
        if (v <= 0)
            continue;
        end
        Z = distances(2) / sqrt(v);
        X = x * Z;
        Y = y * Z;

        lengths = [lengths; X Y Z];
    end

end
% do normalization for preparing P3P
function [mu, mv, mk] = p3pNorm(mu, mv, c_x, c_y, f_x, f_y)
    mu = (mu - c_x) / f_x;
    mv = (mv - c_y) / f_y;
    norm = sqrt(mu * mu + mv * mv + 1);
    mk = 1 / norm;
    mu = mu / norm;
    mv = mv / norm;
end
```

B. Corner Function

```matlab
% use corner function to get corner points, do homogenization
function corners = get_corners(fig)
    corners = corner(fig);
    [corner_m corner_n] = size(corners);
end
```

C. Function for Finding Optimal Rotation and Translation

```matlab
% This function finds the optimal Rigid/Euclidean transform in 3D space
% It expects as input a Nx3 matrix of 3D points.
% It returns R, t

% You can verify the correctness of the function by copying and pasting these commands:


R = orth(rand(3,3)); % random rotation matrix

if det(R) < 0
    V(:,3) = -1*V(:,3);
    R = V*U';
end

t = rand(3,1); % random translation

n = 10; % number of points
A = rand(n,3);
B = R*A' + repmat(t, 1, n);
B = B';

[ret_R, ret_t] = rigid_transform_3D(A, B);

A2 = (ret_R*A') + repmat(ret_t, 1, n);
A2 = A2';

% Find the error
err = A2 - B;
err = err .* err;
err = sum(err(:));
rmse = sqrt(err/n);

disp(sprintf('RMSE: %f', rmse));
disp('If RMSE is near zero, the function is correct!');



% expects row data
function [R,t] = rigid_transform_3D(A, B)
    if nargin ~= 2
        error('Missing parameters');
    end

    %assert(size(A) == size(B))

    centroid_A = mean(A);
```

```
    centroid_B = mean(B);

    N = size(A,1);

    H = (A - repmat(centroid_A, N, 1))' * (B - repmat(centroid_B, N, 1));

    [U,S,V] = svd(H);

    R = V*U';

    if det(R) < 0
        V(:,3) = -1 * V(:,3);
        R = V*U';
    end

    t = -R*centroid_A' + centroid_B';
end
```