

EE 5450 Project 01: Simultaneous Monocular Calibration and Pose Estimation

David R. Mohler

April 17, 2018

1 Introduction

Determination of the pose of rigid objects within images is a common problem within many practical application of computer vision, robotics, computer graphics, etc. The following set of experiments are focused on the application of the simultaneous monocular calibration and pose estimation algorithm to a series of images of rigid objects with known dimensions and correspondence points. From this known data, the algorithm is capable of calibrating a single monocular camera with a fixed focal length. The calibration enables simultaneous discovery of the pose of the viewed object, and from this generate a three dimensional representation of that same object.

2 Methods and Results

The foundation of this project is based on the known dimensions of a given rigid object, a box for example, which is manually assigned a number of points on the object corresponding to its corners (i.e. the coordinates in the object frame represent the dimensions of the object). A model of the initial box and its correspondence points can be seen in Figure 1, additionally, the measured dimensions of the object are shown in Table 1. Given this data we are able to proceed with the implementation of the monocular pose algorithm.

Using the object coordinates in their homogeneous form (i.e. $X_o^i = [x_o^i \ y_o^i \ z_o^i \ 1]^T$), we begin with calculating an approximation of the projection matrix, Π_{est} . The projection matrix is such that $\Pi = [KR \ KT] \in \Re^{3 \times 4}$, where K is the camera calibration matrix, R is a rotation matrix, and T is the translation vector. Given that we know the location of the correspondence points, χ^{pj} , and their matching locations in the object frame, X_0^j , assuming that a sufficient number of points are provided, we are then able to find an approximation to Π . This approximation can be expressed as a least squares minimization of Equation 1, where e_3 is the standard basis vector $[0 \ 0 \ 1]^T$, Π^S is the vectorized or “stacked” version of Π , and \otimes represents the Kronecker product. The least squares estimate of the vectorized projection matrix can be found as the minimum input direction of N , which is the final right singular vector yielded from the singular value decomposition (SVD) of N .

$$N^j \Pi^{pj} = [(X_o^{jT} \otimes I_3) - (X_o^{jT} \otimes \chi^{pj} e_3^T)] \Pi^S = 0 \quad (1)$$

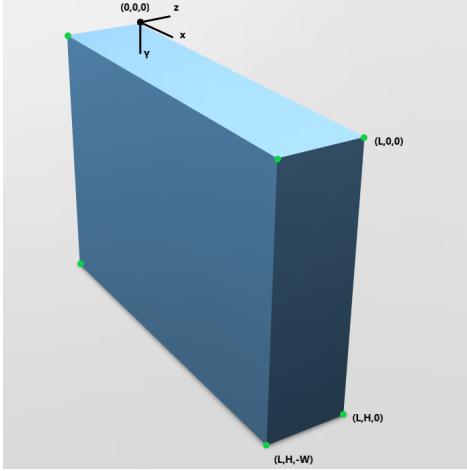


Figure 1: Rigid object model.

Parameter	Value (cm)
Length	45.6
Height	32.5
Width	10.1

Table 1: Object Dimensions

From this we de-vectorize the result of the SVD and obtain Π_{est} . Using this, all necessary components to describe the calibration and pose of the camera are extracted. From order reversing the standard QR decomposition we are able to obtain the scaling factor α , the calibration matrix, and the rotation matrices corresponding to the given set of pixel coordinates. Lastly we retrieve the translation vector associated with the system from Equation 2, where $T' = KT$, and is the rightmost column of Π_{est} :

$$T = \frac{K^{-1}T'}{\alpha} \quad (2) \qquad \chi^{pj} = \frac{\Pi_{est}X_0^j}{\lambda^j} \quad (3) \qquad X_o^{pj} = R^T K^{-1}(\lambda\chi^p - KT) \quad (4)$$

Using the information obtained from the algorithm, we next show the ability to project estimated pixel coordinates in to the image plane. These estimated pixel coordinates are described by Equation 3. From this we are able to qualitatively visualize the success of the algorithm in matching provided correspondence points (Figure 2). In order to do so, we apply the algorithm to four different images of the same object (See Appendix A), all taken from separate locations, thus enabling the ability to test the algorithm against multiple view points. From the estimated pixel coordinates in noiseless reconstructions of the object, using Equation 4, we calculate the average root mean square error (RMSE) of the position between the true coordinates and their respective estimates in the object frame in order to establish the overall fidelity of the algorithm across multiple images taken with varying rotations, translations, and calibrations. In this case PRMSE is expressed by Equations 5 and 6, where n_{im} is the number of images, and n_f is the number of features used:

$$d_i = \sqrt{(x_o^{ij} - \hat{x}_o^{ij}) + (y_o^{ij} - \hat{y}_o^{ij}) + (z_o^{ij} - \hat{z}_o^{ij})} \quad \text{for } j = 1, 2, \dots, n_{im} \quad (5)$$

$$PRMSE = \frac{\sum_{j=1}^{n_{im}} \sqrt{\sum_{i=1}^{n_f} \sqrt{di/n_f}}}{n_{im}} \quad (6)$$

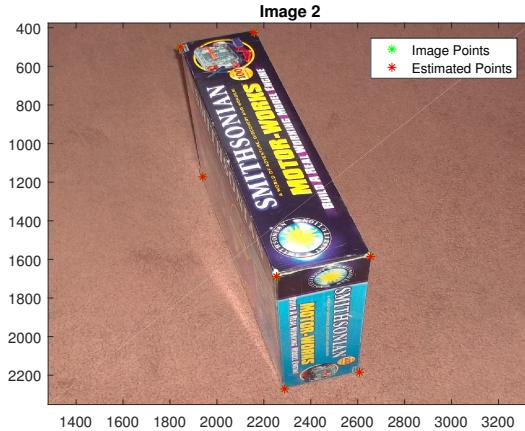


Figure 2: Estimated pixel coordinates

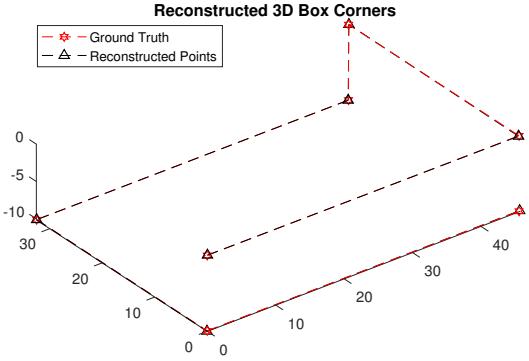


Figure 3: Reconstructed box relative to ground truth

Across the four tested images (with no noise corruption) through 100 iterations of the algorithm, the average distance error was relatively small at $PRMSE = 0.2763 \text{ cm}$.

2.1 Corrupted Correspondence Points

Once the results of the algorithm are proven to work on its own data sets we observe the effect of the calibration and pose established by the algorithm on the pure data set (as opposed to the estimated) data to observe the distortions due to noise or error in correspondence. When applying normally distributed to each component of the correspondence points individually with a standard deviation of $\sigma = 100$ pixels we receive reconstruction results similar to those seen in Figure 4. From the calibration received, we test the ability to reconstruct the true (uncorrupted) data. This yields considerable disfigurement of the object. The reconstruction of the original box when corrupted by noise and the ground truth can be seen in Figure 5. To quantitatively capture the results of the noise reconstruction we compare the angles created between the correspondence points. Since the object is a box and the correspondence points lie along the edges, it is expected that all adjacent vectors are orthogonal to each other. For example, a representative data set of angles associated with the three visible faces of the box, and known correspondences appears, in Table 2 demonstrating the angle deviations due to noise in the correspondence points.

Point	Angle(s) (degrees)
1	103.6447
2	82.6421, 90.4116
3	88.9373
4	91.496, 92.3435
5	93.2852
6	85.5876, 76.0264
7	97.667, 88.7342, 89.1479

Table 2: Object Dimensions



Figure 4: Estimated image coordinates (unique coordinate component noise)

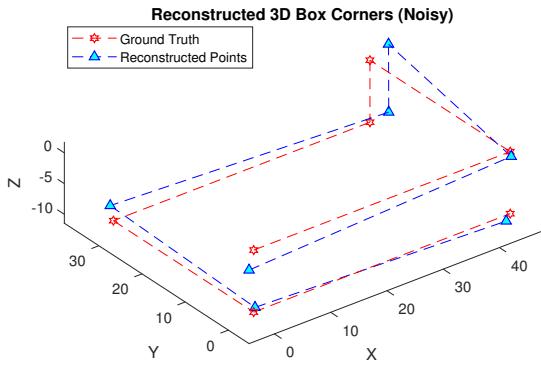


Figure 5: Reconstructed box relative to ground truth

As would be expected, the angles are centered about the appropriate mean value of 90 degrees, however the deviations from precisely right angles by as much as 15.53 degrees shows the obvious loss of the shape of the true object upon reconstruction. Similarly, the reliability of the reconstruction is tied to the amount of noise introduced in to the captured pixel coordinates. Under the condition of $\sigma = 100$ pixels, using 100 successive trials of the algorithm, there is an average PRMSE introduced of 1.65 cm, or an increase of error by approximately 595.4%. Through observation of various noise levels we observe that the RMS position error is (nearly) linear with the standard deviation of the noise (Figure 6). It is essential to note that beyond a standard deviation of approximately 200 pixels it becomes common for pixels to be projected in to non-existent pixel mappings, and as such, are obviously invalid results.

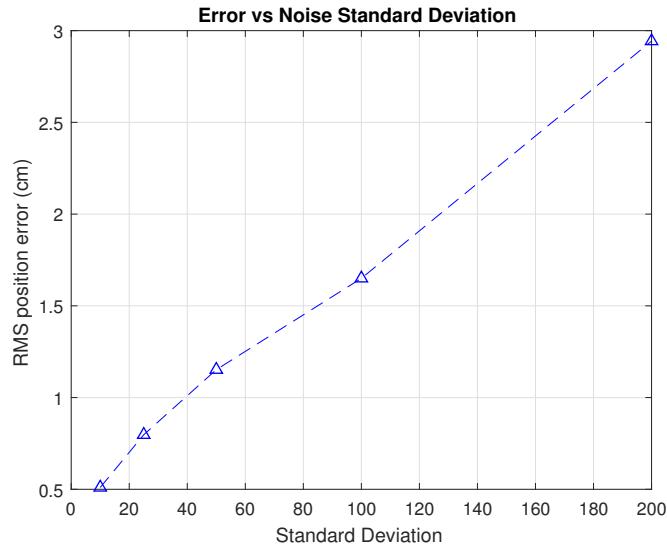


Figure 6: RMS Error VS. standard deviation of applied noise

2.2 Improper Dimensions

In order for the monocular pose algorithm to appropriately calibrate and successively reconstruct three dimensional objects it assumes the a priori knowledge of an object within the image's complete dimensions. To observe the effect of the error that is caused when incorrect dimensions are given to the algorithm, the true dimensions of the box were altered. This produced concerning results. By increasing the width of the box by a factor of 10 to 101, we are able to observe the pixel coordinate projection in Figure 7. Given only the image coordinate

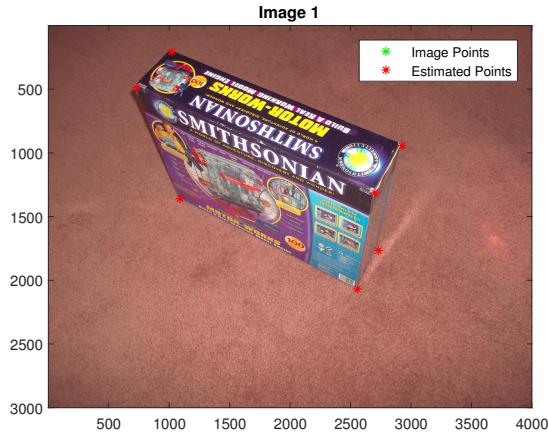


Figure 7: Estimated image coordinates
(inaccurate dimensions)

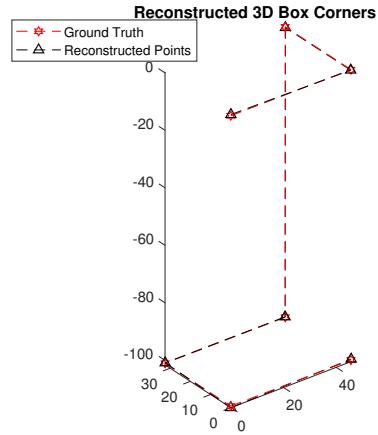


Figure 8: Reconstructed box relative to
ground truth (inaccurate dimensions)

projections it could easily be assumed that the algorithm had appropriately reconstructed the box, despite the wrong dimensions. However, we then project these pixel coordinates and find that in the object frame, the computer believes that the reconstruction should match the dimensions that were provided by the user (see Figure 8). Numerical evidence lies within the translation vector, T , yielded by the monocular pose algorithm. For instance, using the initial given image (Figure 4) it can be found that $T = [-21.8 \quad -33.8 \quad 90.0]^T$ cm which has a length of 98.61 centimeters. In order for the algorithm to provide a reconstruction of the box where the width is falsely increased by a factor of ten that the translation vector is $T = [507.8 \quad -121.1 \quad 99.2]^T$ which has a length of 531.4cm. This implies that without the true box having been moved in the image, the translation vector implies that the origin of the box is an additional 4.32 meters farther away than it is in reality. As such, this algorithm, has potential for exceedingly poor calibration and misleading visual results in the event that objects being used to calibrate the system are not properly measured.

3 Conclusions

In this project we have demonstrated the ability to compute the necessary information for the simultaneous calibration and pose estimation of a single monocular camera. The algorithm itself proved to be a fairly straightforward implementation, and in ideal or near ideal environments is able to provide reasonable estimates of the calibration of the camera,

the object pose, and associated depths to the known correspondence points. However, this algorithm is not without its own problems. In the presence of high levels of noise corrupting the correspondence points a near linear increase in 3D position error is shown, and as a result the orthogonality of the box reconstruction is compromised beyond reasonable tolerances. Also, we have shown that in the event of inaccurate measurements of calibration objects being delivered to the algorithm that it is incapable of providing accurate scene reconstructions as well as yielding misleading images that are poorly indicative of the error that has occurred.

A Additional Images

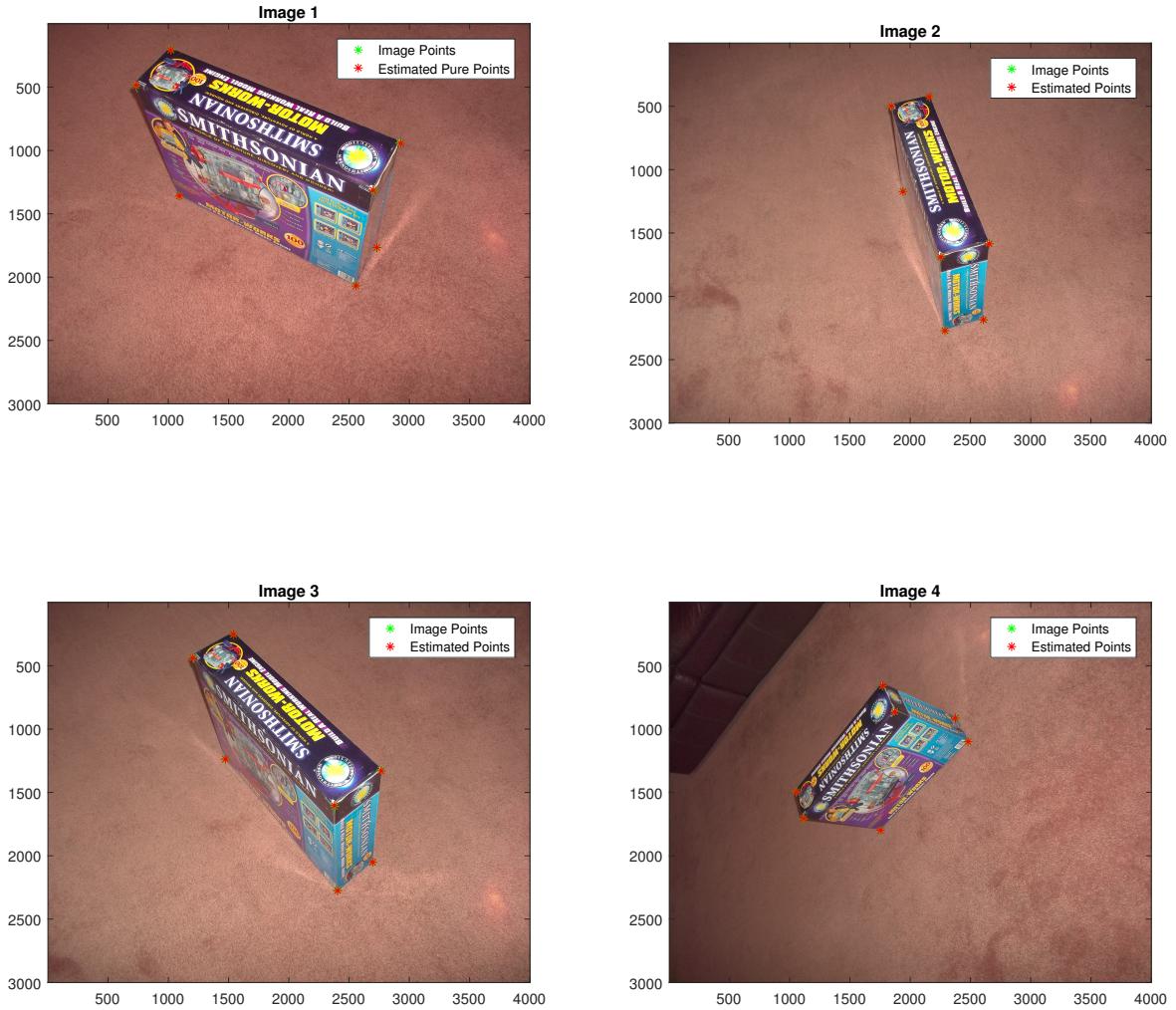


Figure 9: Four tested images.

B Code Listings

Listing 1: Top level implementation for Monocular Calibration and Pose Estimation

```
1 %David R Mohler
2 %EE-5450: Topics in Robotics
3 %Project 1
4 %Spring 2018
5
6 clear
7 close all
8
9 meanvec = zeros(100,1);
10 angmean = meanvec;
11
12 %read in the images
13 i1=imread('Images\cvClass_023.jpg','jpg');
14 i2=imread('Images\cvClass_026.jpg','jpg');
15 i3=imread('Images\cvClass_024.jpg','jpg');
16 i4=imread('Images\cvClass_028.jpg','jpg');
17 lbox=45.6; %length of box (assume centimeters)
18 hbox=32.5; %height of box
19 wbox=10.1; %width of box
20 n=7;
21
22 %use either the first set of commands (to initially mark
23 %    [+]) correspondence
24 %points manually) or the load command (to read in previous
25 %    [+]) correspondence
26 %points)
27 %
28 %    X1=[1 1];X2=[1 1];%must start with an initial point; this
29 %    [+]) will be removed later
30 %    [X1,X2]=cpselect(i1,i2,X1,X2,'Wait',true);
31 %    save motorBoxCorners23_26 X1 X2 %save the correspondence
32 %    [+]) points you just found
33
34 %load in correspondence points
35 load motorBoxCorners23_26.mat
36 load motorBoxCorners24_28.mat
37
38 [mc,nc]=size(X1); %need if changing # of Correspond. pts.
39 X1=X1(2:n+1,:); %remove initial point, it is not good data
40 X2=X2(2:n+1,:);
41 [mc,nc]=size(X1);
```

```

x1pixmat=[X1 '
39      ones(1,mc)]; %convert the points to homogeneous coordinates
x2pixmat=[X2 '
41      ones(1,mc)]; %convert the points to homogeneous coordinates
x3pixmat=[X3 '
43      ones(1,mc)]; %convert the points to homogeneous coordinates
x4pixmat=[X4 '
45      ones(1,mc)]; %convert the points to homogeneous coordinates

47 % for j = 1:100
    %Corruption of data points with gaussian noise
49 x1pixcor = x1pixmat;
    %corruption of each coordinate in each point seperately
51 for i = 1:mc
        x1pixcor(1,i) = x1pixmat(1,i)+100*randn(1,1);
53        x1pixcor(2,i) = x1pixmat(2,i)+100*randn(1,1);
end

55
%Corrupting all coordinates equally across correspondence
    [+]points
57 %(maintains shape, but is shifted in image plane)
    % x1pixcor(1,:) = x1pixmat(1,:)+75*randn(1,1);
59    % x1pixcor(2,:) = x1pixmat(2,:)+75*randn(1,1);
    % x1pixcor(3,:) = x1pixmat(3,:)+75*randn(1,1);

61
%enter object coordinates for the first 7 corners
63 Xomat=[0 lbox lbox lbox 0 0 lbox
            0 0 hbox hbox hbox 0 0
            0 0 0 -wbox -wbox -wbox -wbox]; %object coords of the
65                [+]four corners
Xoh = [Xomat; ones(1,mc)]; %Homogeneous object coordinates
67 %find calibration matrices
[gest1qr,lambda1qr,K1]=monoPoseQR(Xomat,x1pixmat); %find K,
    [+]depth, and g
69 Rest1qr=gest1qr(1:3,1:3);Test1qr=gest1qr(1:3,4); %Extraction
    [+]of R and T

71 %cor indicates corrupted data
[gest1cor,lambda1cor,K1cor]=monoPoseQR(Xomat,x1pixcor);
73 Rest1cor=gest1cor(1:3,1:3);Test1cor=gest1cor(1:3,4);

75 [gest2qr,lambda2qr,K2]=monoPoseQR(Xomat,x2pixmat);
Rest2qr=gest2qr(1:3,1:3);Test2qr=gest2qr(1:3,4);

77 [gest3qr,lambda3qr,K3]=monoPoseQR(Xomat,x3pixmat);

```

```

79 Rest3qr=gest3qr(1:3,1:3);Test3qr=gest3qr(1:3,4);

81 [gest4qr,lambda4qr,K4]=monoPoseQR(Xomat,x4pixmat);
Rest4qr=gest4qr(1:3,1:3);Test4qr=gest4qr(1:3,4);

83 PI1 = [K1*Rest1qr K1*Test1qr];
85 PI1cor = [K1cor*Rest1cor K1cor*Test1cor];
87 PI2 = [K2*Rest2qr K2*Test2qr];
PI3 = [K3*Rest3qr K3*Test3qr];
PI4 = [K4*Rest4qr K4*Test4qr];

89 %Estimated pixel coordinates of correspondence points
91 x1pixest = zeros(size(x1pixmat));
x1pixestcor = zeros(size(x1pixmat));
93 x2pixest = zeros(size(x2pixmat));
x3pixest = zeros(size(x3pixmat));
95 x4pixest = zeros(size(x4pixmat));
96 for i = 1:mc
97     x1pixest(:,i) = (PI1*Xoh(:,i))/lambda1qr(i);
98     x1pixestcor(:,i) = (PI1cor*Xoh(:,i))/lambda1cor(i);
99     x2pixest(:,i) = (PI2*Xoh(:,i))/lambda2qr(i);
100    x3pixest(:,i) = (PI3*Xoh(:,i))/lambda3qr(i);
101    x4pixest(:,i) = (PI4*Xoh(:,i))/lambda4qr(i);
102 end

103 %Reconstruction of Object Coordinates from estimates
105 Xoest1 = zeros(size(Xomat));
Xoest1cor = zeros(size(Xomat));
107 Xoest2 = zeros(size(Xomat));
Xoest3 = zeros(size(Xomat));
109 Xoest4 = zeros(size(Xomat));
110 for i = 1:mc
111     Xoest1(:,i) = Rest1qr'*inv(K1)*(lambda1qr(i)*x1pixmat(:,i)-
112         [+]K1*Test1qr);
113     Xoest1cor(:,i) = Rest1cor'*inv(K1cor)*...
114         (lambda1cor(i)*x1pixmat(:,i)-K1cor*-
115             [+]Test1cor);
116     Xoest2(:,i) = Rest2qr'*inv(K2)*(lambda2qr(i)*x2pixmat(:,i)-
117         [+]K2*Test2qr);
118     Xoest3(:,i) = Rest3qr'*inv(K3)*(lambda3qr(i)*x3pixmat(:,i)-
119         [+]K3*Test3qr);
120     Xoest4(:,i) = Rest4qr'*inv(K4)*(lambda4qr(i)*x4pixmat(:,i)-
121         [+]K4*Test4qr);
122 end

```

```

119 %Mean Distance PRMSE
dist1 = sqrt((Xomat(1,:)-Xoest1(1,:)).^2+(Xomat(2,:)-Xoest1
    [+] (2,:)).^2 ...
121 +(Xomat(3,:)-Xoest1(3,:)).^2);
dist2 = sqrt((Xomat(1,:)-Xoest2(1,:)).^2+(Xomat(2,:)-Xoest2
    [+] (2,:)).^2 ...
123 +(Xomat(3,:)-Xoest2(3,:)).^2);
dist3 = sqrt((Xomat(1,:)-Xoest3(1,:)).^2+(Xomat(2,:)-Xoest3
    [+] (2,:)).^2 ...
125 +(Xomat(3,:)-Xoest3(3,:)).^2);
dist4 = sqrt((Xomat(1,:)-Xoest4(1,:)).^2+(Xomat(2,:)-Xoest4
    [+] (2,:)).^2 ...
127 +(Xomat(3,:)-Xoest4(3,:)).^2);
avDRMSE = (sqrt(sum(dist1/mc))+sqrt(sum(dist1/mc))+sqrt(sum(
    [+] dist1/mc))...
129 +sqrt(sum(dist1/mc)))/4;

131 dist1cor = sqrt((Xomat(1,:)-Xoest1cor(1,:)).^2+(Xomat(2,:)-
    [+] Xoest1cor(2,:)).^2 ...
133 +(Xomat(3,:)-Xoest1cor(3,:)).^2);
DRMSEcor = sqrt(sum(dist1cor/mc));
%
%meanvec(j) = DRMSEcor;

135 %construct vectors between known vertices to check angles
137 v12 = Xoest1cor(:,2)-Xoest1cor(:,1);
v16 = Xoest1cor(:,6)-Xoest1cor(:,1);
139 v21 = -v12;
v23 = Xoest1cor(:,3)-Xoest1cor(:,2);
141 v27 = Xoest1cor(:,7)-Xoest1cor(:,2);
v32 = -v23;
143 v34 = Xoest1cor(:,4)-Xoest1cor(:,3);
v47 = Xoest1cor(:,7)-Xoest1cor(:,4);
145 v43 = -v34;
v45 = Xoest1cor(:,5)-Xoest1cor(:,4);
147 v54 = -v45;
v56 = Xoest1cor(:,6)-Xoest1cor(:,5);
149 v61 = -v16;
v65 = -v56;
151 v67 = Xoest1cor(:,7)-Xoest1cor(:,6);
v72 = -v27;
153 v74 = -v47;
v76 = -v67;

155 %Extract Angles b/w vectors
157 Theta216 = atan2d(norm(cross(v16,v12)),dot(v16,v12));

```

```

159 Theta127 = atan2d(norm(cross(v27,v21)),dot(v27,v21));
Theta327 = atan2d(norm(cross(v27,v23)),dot(v27,v23));
Theta234 = atan2d(norm(cross(v32,v34)),dot(v32,v34));
Theta347 = atan2d(norm(cross(v43,v47)),dot(v43,v47));
Theta547 = atan2d(norm(cross(v45,v47)),dot(v45,v47));
Theta456 = atan2d(norm(cross(v54,v56)),dot(v54,v56));
Theta567 = atan2d(norm(cross(v65,v67)),dot(v65,v67));
Theta167 = atan2d(norm(cross(v61,v67)),dot(v61,v67));
Theta672 = atan2d(norm(cross(v72,v76)),dot(v72,v76));
Theta674 = atan2d(norm(cross(v74,v76)),dot(v74,v76));
Theta724 = atan2d(norm(cross(v72,v74)),dot(v72,v74));

169 faceAngles = [Theta216 Theta127 Theta327 Theta234 Theta347
    +]Theta547 ...
171     Theta456 Theta567 Theta167 Theta672 Theta674 Theta724];

173 %Print the associated angles to command window (below)
174 ang1 = [ 'Point_1_angles:',num2str(faceAngles(1))];
175 ang2 = [ 'Point_2_angles:',num2str(faceAngles(2)),',',...
    ,num2str(faceAngles(3))];
176 ang3 = [ 'Point_3_angles:',num2str(faceAngles(4))];
177 ang4 = [ 'Point_4_angles:',num2str(faceAngles(5)),',',...
    ,num2str(faceAngles(6))];
178 ang5 = [ 'Point_5_angles:',num2str(faceAngles(7))];
179 ang6 = [ 'Point_6_angles:',num2str(faceAngles(8)),',',...
    ,num2str(faceAngles(9))];
180 ang7 = [ 'Point_7_angles:',num2str(faceAngles(10)),',',...
    ,num2str(faceAngles(11)),',',',',num2str(faceAngles(12))];
181 ang = [ang1 ang2 ang3 ang4 ang5 ang6 ang7];
182 disp(ang1)
183 disp(ang2)
184 disp(ang3)
185 disp(ang4)
186 disp(ang5)
187 disp(ang6)
188 disp(ang7)

193 % end
194 % avg = sum(meanvec)/100
195 %Images and plotting
196 figure(1)
197 clf
198 image(i1);
199 hold on
200 plot(X1(:,1)+j*X1(:,2), 'g*')

```

```

scattered(x1pixest(1,:),x1pixest(2,:),'r*') %Pure pixel estimates
203 %Noise corrupted pixel est.
scattered(x1pixestcor(1,:),x1pixestcor(2,:),'c^','MarkerFaceColor',...
    '[+]c')
205 legend('Image\u20d7Points','Estimated\u20d7Pure\u20d7Points','Corrupted\u20d7Estimate',...
    '[+]')
title('Image\u20d71')
207 hold off

209 figure(2)
clf
211 image(i2)
hold on
213 plot(X2(:,1)+j*X2(:,2),'g*')
scattered(x2pixest(1,:),x2pixest(2,:),'r*')
215 legend('Image\u20d7Points','Estimated\u20d7Points')
title('Image\u20d72')
217 hold off

219
220 figure(3)
clf
221 image(i3)
hold on
222 plot(X3(:,1)+j*X3(:,2),'g*')
223 scattered(x3pixest(1,:),x3pixest(2,:),'r*')
legend('Image\u20d7Points','Estimated\u20d7Points')
225 title('Image\u20d73')
hold off

227
228 figure(4)
clf
229 image(i4)
hold on
230 plot(X4(:,1)+j*X4(:,2),'g*')
231 scattered(x4pixest(1,:),x4pixest(2,:),'r*')
legend('Image\u20d7Points','Estimated\u20d7Points')
233 title('Image\u20d74')
hold off

235
236 figure(5)
hold on
clf
237 pts=1:7;
plot3(Xomat(1,pts),Xomat(2,pts),Xomat(3,pts),'--rh')

```

```

245 axis equal
    title('3D_Box_Corners_(Ground_Truth)')
247 hold off

249 figure(6)
hold on
clf
pts=1:7;
253 plot3(Xoest1(1,pts),Xoest1(2,pts),Xoest1(3,pts), '--g*')
axis equal
255 title('Reconstructed_3D_Box_Corners')
hold off

257 %Reconstruction of truth and noise boxes
259 figure(7)
clf
261 pts=1:7;
hold on
263 plot3(Xomat(1,pts),Xomat(2,pts),Xomat(3,pts), '--rh')
plot3(Xoest1(1,pts),Xoest1(2,pts),Xoest1(3,pts), '--k^')
265 view(3)
axis equal
267 title('Reconstructed_3D_Box_Corners_From_True_Data')
legend('Ground_Truth','Reconstructed_Pure_Points')
269 hold off

271 figure(8)
clf
273 pts=1:7;
hold on
275 plot3(Xomat(1,pts),Xomat(2,pts),Xomat(3,pts), '--rh')
plot3(Xoest1cor(1,pts),Xoest1cor(2,pts),Xoest1cor(3,pts), '--b^',...
      , 'MarkerFaceColor', 'c')
277 view(3)
axis equal
title('Reconstructed_3D_Box_Corners_(Noisy)')
281 legend('Ground_Truth','Reconstructed_Points')
hold off

```

Listing 2: QR Decomp Based Algorithm Implementation

```

function [g,lambda,K]=monoPoseQR(Xomat,xpixh)
2 %Function to find the camera pose (g), depth to object points (
%     [+]lambda), and
%camera calibration matrix (K)
4

```

```

%David R Mohler
6 %EE-5450 Fall 2018

8 n = size(Xomat,2);

10 %convert object coords to homogeneous coordinates
Xoh = [Xomat; ones(1,n)];
12 I3 = eye(3);
e3 = [0 0 1]';

14 N = zeros(3*n,12);
16 j = 1;
for i = 1:n
18 N(j:j+2,:) = kron(Xoh(:,i)',I3)-kron(Xoh(:,i)',(xpixh(:,i)*e3
    [+]'));
j = j+3;
20 end

22 [~,~,V] = svd(N); %Use only Right singular vectors
PIs = V(:,end); %Extract LSE of stacked PI
24 PI = reshape(PIs,[3 4]); %Unstack PI vector
[alf,K,R] = qrCommute(PI(:,1:3)); %use QR decomp to find K,R, and
    [+]scale
26 T = (inv(K)/alf)*PI(:,4); %Calculate translation
PIest = [K*R K*T];
28 g = [R T;0 0 0 1]; %Construct pose matrix
lambda = e3'*PIest*Xoh; %Extract depth

```

Listing 3: Rearranged QR Decomposition (Credit: Dr. John McInroy)

```

1 function [alf, K,R ] = qrCommute( M )
%function [alf, K,R ] = qrCommute( M )

3
%modified qr decomposition. It commutes the order
5 %of the rotation matrix and the upper triangular matrix.
%
7 %The decomposition is returned such that alf*K*R = M, where
%alf is a scalar, K is upper triangular, and R is a special
    [+]orthogonal
9 %matrix.
%
11 %
%In addition, it makes the diagonal of K positive, with the last
    [+]entry
13 %of K, K(n,n), equal to one. This factorization is ideal for
    [+]uncalibrated

```

```

%camera factorization problems. If it is not possible, an error
[+] is
15 %returned.
%
17 %
%J. McInroy, 2/4/10
19
21 [m,n]=size(M);
E=zeros(n,n);
E2=zeros(n,n);
23 for i=1:n,
    E(i,n+1-i)=1;
25 end

27 [Q,Rp]=qr(M'*E);
Kn=E*Rp';
29 Rn=Q';
alf=Kn(n,1);
31 for i=1:n,
    if Kn(i,n+1-i)<0,
        E2(n+1-i,i)=-1;
    else
        E2(n+1-i,i)=1;
    end
37 end

39 R=E2'*Rn;
K=Kn*E2/alf;
41 if det(R)<0,
    R=-R;
43     alf=-alf;
end
45 if K(n,n)<0,
    K=-K;
47     alf=-alf;
end
49

51 end

```