ENGN4528/6528 Computer Vision – 2015 Computer-Lab 5 (C-Lab5)

Sai Ma - u5224340

May 2015

1 Camera Calibration

In this task, we asked to develop a method to calibrate camera, finding the relationship between 3D world coordiantes and their 2D projection positions in the image. In this task, we needs two sets of points. We select the 2D coordinate points on a testing image (e.g. *stereo2012a.jpg*) by Matlab method *ginput()*. At the same time, we construct a 3D coordiante points information based on given image displayed information. The calibration target image has three mutually orthogonal faces with marked points. These points are located on the regular grid, and apart from 7 cm. Therefore, we construct *original* point coordinates in these grids. We can build the 3D system as figure 1 shown.

Therefore, I create 12 3D points in this coordinate system, and select their corresponding image location to get their 2D points coordinates. In order to use them easily, I also use Matlab method *load* and *save* to save and load these arguments.

```
1 % for the XYZ, it is constructed by ourselve, we use same ...
     XYZ system in two
2 % different images
3 XYZ = [0,7,7; 0,7,14; 0,14,14; 0,14,7; 0,21,7; 0,21,14;
4         7,7,0; 14,7,0; 14,14,0; 7,14,0; 7,21,0; 14,21,0];
5
6 % read image, and build uv system
7 imageName1 = 'stereo2012a.jpg';
8 img = imread(imageName1);
```

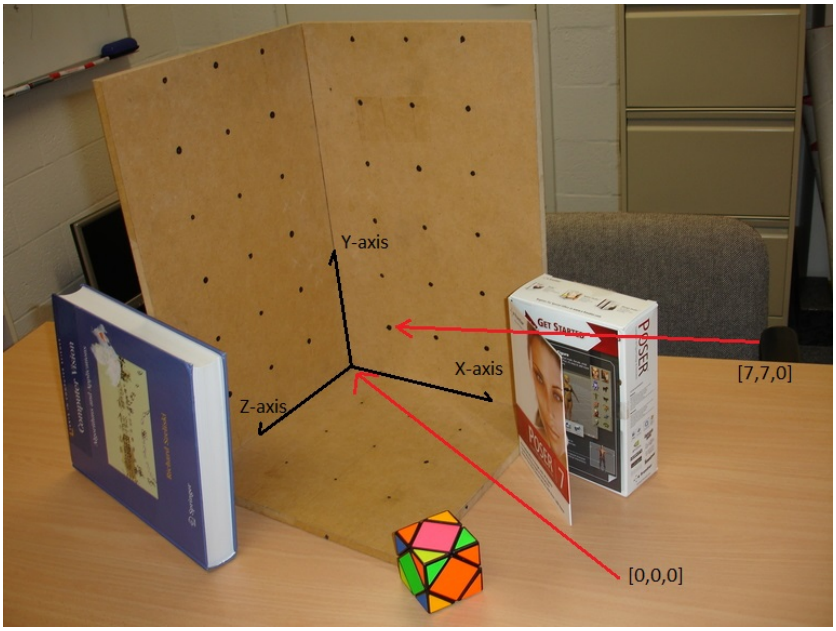

Figure 1: Stereo2012a With 3D Coordinate System

```
9
10 figure('name', imageName1);
11 imshow(img);
12 hold on
13
14 % select 12 points from this image
15 [u1, v1] = ginput(12);
16
17 change uv to input style
18 uv = [u1, v1];
19 save('stereo2012a.mat', 'uv');
20
21 load('stereo2012a.mat');
```

Then, the next job is to construct method named *calibrate()*. Its inputs are 3D coordinate points, 2D coordinate points and image. As the lecture taught, we use 3D points XYZ and 2D points uv to construct these n 2×12 matrices A_i, then merge into a single $2n \times 12$ matrix A. Then we compute SVD of A, then get our solution P at last column of V.

```
%In this method, the main task is constract matrix A, then ...
    get p
[pointNum, ¬] = size(uv);

A = zeros(pointNum*2, 12);

for AIndex = 1 : pointNum

    X = XYZ(AIndex, 1);
    Y = XYZ(AIndex, 2);
    Z = XYZ(AIndex, 3);

    u = uv(AIndex, 1);
    v = uv(AIndex, 2);

    A(AIndex*2 - 1, :) = [X, Y, Z, 1, 0, 0, 0, 0, -u*X, ...
        -u*Y, -u*Z, -u];
    A(AIndex*2, :) = [0, 0, 0, 0, X, Y, Z, 1, -v*X, -v*Y, ...
        -v*Z, -v];

end

[¬, ¬, V] = svd(A,0);
P = reshape(V(:,end), 4, 3)';
```

Then, we get our target P. In order to prove that we get the correct P, I plan to multiply 3D points by P, and get transformed uv points. At the same time, we can calculate mean square error between uv and $uvTrans$. Then, we plot original and transformed points on the image to compare distances between these points.

```
% then, write a method to get reprojection error, get our ...
    transformed
% points
XYZone = [XYZ, ones(pointNum, 1)]'; % construct XYZone ...
    with four columns
uvTrans = P*XYZone; % get transformed uv values
uvTrans(1,:) = uvTrans(1,:)./uvTrans(3,:); % make ...
    transformed uv third row is 1
uvTrans(2,:) = uvTrans(2,:)./uvTrans(3,:);

% get Euclidean distance between re-projected points and ...
    original points
reprojectError = sqrt(sum(sum((uvTrans(1:2,:) - uv').^2)));
```

```
10
11 fprintf('The mean reprojection error is: %d.\n', ...
       reprojectError/pointNum);
12
13 % also show our select points in image, with coordinate ...
       calcuated by P
14 uvTrans = uvTrans';
15
16 figure('Name', 'Selection Points Compare with Calcuated ...
       Points');
17 imshow(im);
18 hold on;
19 plot(uv(:,1),uv(:,2),'g+');
20 plot(uvTrans(:,1),uvTrans(:,2),'ro');
21 hold off;
```

The following figure 2 and figure 3 are these compared results.

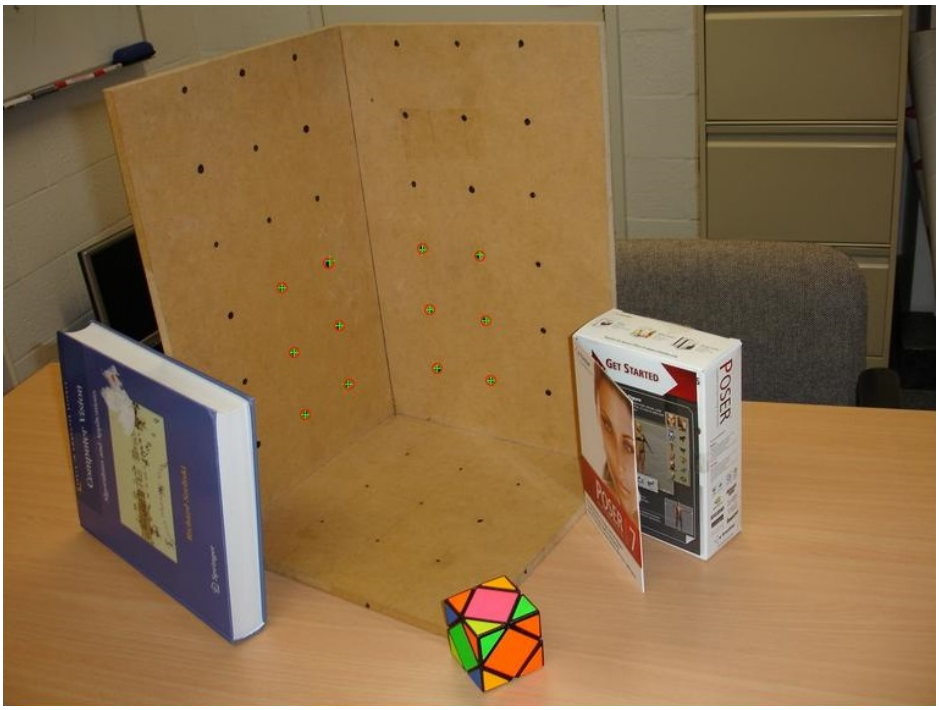

Figure 2: Stereo2012a Original Points and Transformed Points

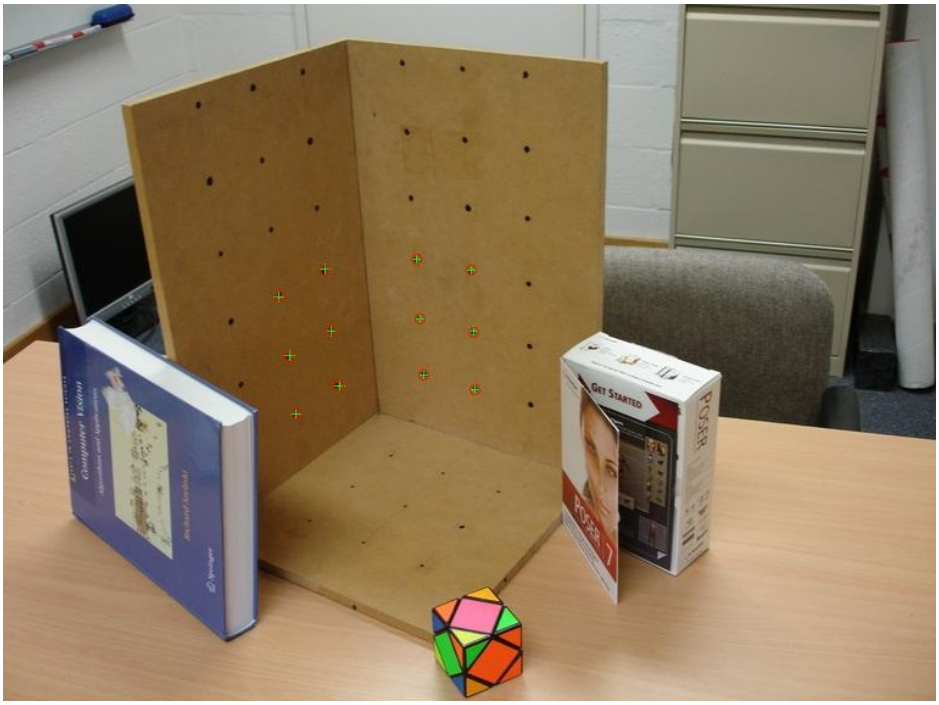

Figure 3: Stereo2012b Original Points and Transformed Points

1.1 Q1: List the Basic Derivation of DLT algorithm

In the DLT algorithm, we have 2D and 3D point pairs, and we aim to get P where $x_i = PX_i$. In the equation, the x_i mean the i 2D coordinates, and the X_i mean i 3D coordinate come from point pairs. The x is 2D point, and we change it to $(x, y, w)^T$. Consider we can get $x_i \times PX_i = 0$ from original equation. Then we construct the relationship 1.1 that:

$$x_i \times PX_i = \begin{bmatrix} y_i p_3^T X_i - w_i p_2^T X_i \\ w_i p_1^T X_i - x_i p_3^T X_i \\ x_i p_2^T X_i - y_i p_1^T X_i \end{bmatrix}$$

where $[p_i, p_2, p_3]^T$ equals to P_i.

When we perform factorize unknowns arguments, we get the equation 1.1 is:

$$\begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0^T & -x_i X_i^T \\ -y_i X_i^T & x_i X_x^T & 0^T \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = A_i p = 0$$

In this problem, we only need first two line of equations to solve the p. Then we aims to solve the equation 1.1

$$\begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0^T & -x_i X_i^T \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = A_i p = 0$$

This matrix A has 11 degree of freedom, and we need 6 pairs of 2D and 3D points to solve this p, and its solution is 1D nullspace from singular value decomposition of matrix A.

1.2 Q2: Minimally Points and Mean Squared Error

In our target P, it is a 3×4 matrix and defines as:

$$\begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix}$$

And once we divide p by p_{23}, and this matrix has 11 unknown arguments. As we known, the 11 degree of freedom matrix require 11 point coordinates to solve, and which means 6 pairs of 2D and 3D point coordinates. As a result, in order to solve this P in 3D DLT, we require at least 6 pairs of 2D and 3D points.

In the image *stereo*2012*a.jpg*, my solution on p is:

$$\begin{bmatrix} 0.008253 & -0.004482 & -0.013989 & 0.694305 \\ -0.000617e-4 & -0.015833 & 0.001591 & 0.719304 \\ -1.136248e-05 & -8.123602e-06 & -1.610913e-05 & 0.002155 \end{bmatrix}$$

with the mean square error is: 0.189501.

In the image *stereo*2012*b.jpg*, my solution on p is:

$$\begin{bmatrix} 0.008564 & -0.004041 & -0.012489 & 0.677494 \\ 0.000483 & -0.015776 & 0.002773 & 0.735183 \\ -9.514198e-06 & -9.034940e-06 & -1.116055e-05 & 0.002171 \end{bmatrix}$$

with the mean square error is: 0.268517.

1.3 Q3: Decompose P to K, R, t

In the process of decomposition of camera matrix P to K, R and t , we have the relationship on camera projection matrix is:

$$P = KR[I| - \tilde{C}]$$

where C is the camera center and expressed as $C = (\tilde{C}, 1)$, K is the calibration matrix. Then, the calibration matrix can be obtained from a *RQ decomposition* of the first 3×3 sub matrix of the camera matrix P. The reason is that K states as:

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \beta_x & y_0 \\ & & 1 \end{bmatrix}$$

The K is an upper triangular. At the same time, R is the orthogonal rotation matrix between the camera and world coordinate frames. On the other hand, we have a 3×4camera projection matrix P, then:

$$P = [M| - M\tilde{C}] = M[I| - \tilde{C}]]$$

Where M is the first 3×3 sub matrix of P. Compare with first camera projection matrix, we can achieve calibration matrix K and R via RQ decomposition of the first 3×3 sub matrix of the camera matrix P. This can write in Matlab as:

```
N = size(P,1);
M = P(:,1:N);

[K,R] = vgg_rq(M);
```

Where the method named *vgg_rg(H)* perform RQ decomposition of M.

```
function [ K, R ] = vgg_rq( M )
% This method used to calcuated the K and R value based on ...
    given homography
% matrix.

    M = M';
```

```
6       [R, K] = qr(M(end:-1:1,end:-1:1)); % perform ...
           Orthogonal-triangular decomposition.
7
8       R = R';
9
10      R = R(end:-1:1,end:-1:1);
11      K = K';
12
13      K = K(end:-1:1,end:-1:1);
14
15      if det(R)<0
16          K(:,1) = -K(:,1);
17          R(1,:) = -R(1,:);
18      end
19  end
```

At the same time, $P = K * R[eye(3), -t]$. And $K * R$ is the first 3×3 sub-matrix. Then, we have new equation that $P[:, 4] = M * -t$. Therefore, we write Matlab code as:

```
1  t = -P(:,1:N)\P(:,end);
```

Moreover, we need to recover the camera's focal length. we can check image property in operation system, and its get physical focal length. This camera focal length is 8 mm.

In the figure *stereo2012a,jpg*, its K (keep its right-down value is 1)is:

$$\begin{bmatrix} 698.4756172 & 9.234292 & 369.510906 \\ 0 & 706.620628 & 241.9471017 \\ 0 & 0 & 1 \end{bmatrix}$$

The R is:

$$\begin{bmatrix} 0.834251 & -0.087245 & -0.544437 \\ 0.141517 & -0.920446 & 0.364349 \\ -0.532913 & -0.381006 & -0.755537 \end{bmatrix}$$

Then, our next task is to extract rotated angle from R based on the relationship that:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, we extract α, β and γ from R [1]. The α is: -153.23 degrees, β is 32.20 degrees and γ is 9.62 degrees. And the t is $[57.86; 49.99; 67.75]$.

1.4 Q4: List Second Camera Relative Orientation and Translation

In the picture *stere2012b.jpg*, we can extract its t, R as section 1.3 (the two image have same K as we assumed in question). Then, the R is:

$$\begin{bmatrix} 0.790391 & -0.082483 & -0.607023 \\ 0.264951 & -0.847394 & 0.460134 \\ -0.552341 & -0.524518 & -0.647919 \end{bmatrix}$$

Then, we also perform Matlab method *comp_decomp_matrix()* to extract α is -141.00 degrees, β is 33.52 degrees and γ is 18.53 degrees. And the camera located t is $[-0.64; 63.29; 82.59]$. As a result, we can notice when relative to first camera, the second camera rotate angles are change from -153.23 to -141.00 at α, 32.20 to 33.5 at β and 9.62 to 18.53. And camera translated location are change from $[57.86; 49.99; 67.75]$ to $[-0.64; 63.29; 82.59]$.

1.5 Q5: Fundamental Matrix

Fundamental matrix is a 3×3 matrix of rank 2. Therefore, its size is 3×3. Its degree of freedom is 7. We need 7 pairs of points coordinates to estimate this fundamental matrix.

[1] use a Matlab method download from `http://in.mathworks.com/matlabcentral/fileexchange/43907-compose---decompose-3x3-rotation-matrix--comp-decomp-matrix-`

2 Appendix: Matlab Code

```
% this method aims calcated camera calibration (3*4 camera ...
    matrix P)

clc;
clear;

% for the XYZ, it is constructed by ourselve, we use same ...
    XYZ system in two
% different images
XYZ = [0,7,7; 0,7,14; 0,14,14; 0,14,7; 0,21,7; 0,21,14;
       7,7,0; 14,7,0; 14,14,0; 7,14,0; 7,21,0; 14,21,0];

% load XYZ;

% read image, and build uv system
imageName1 = 'stereo2012a.jpg';
img = imread(imageName1);

% imageName2 = 'stereo2012b.jpg';
% img = imread(imageName2);

% figure('name', imageName1);
% imshow(img);
% hold on
%
% % select 12 points from this image
% [u1, v1] = ginput(12);

% change uv to input style
% uv = [u1, v1];
% save('stereo2012a.mat', 'uv');

% figure('name', imageName2);
% imshow(img2);
% hold on

% select 12 points from this image
% [u2, v2] = ginput(12);

% % change uv to input style
% uv = [u2, v2];
```

```
% save('stereo2012b.mat', 'uv');

% % get uv values from stereo2012a.mat
load('stereo2012a.mat');
% load('stereo2012b.mat');

P = calibrate(img, XYZ, uv);

[K, R, t] = vgg_KR_from_P(P, 1);
```

```
function [ P ] = calibrate( im, XYZ, uv )
% CALIBRATE
%
% Function to perform camera calibration
%
% Usage:   P = calibrate(im, XYZ, uv)
%
%   Where:   im  - is the image of the calibration target.
%            XYZ - is a N x 3 array of  XYZ coordinates
%                  of the calibration target points.
%            uv  - is a N x 2 array of the image coordinates
%                  of the calibration target points.
%            P   - is the 3 x 4 camera calibration matrix.
%  The variable N should be an integer greater than or ...
    equal to 6.
%
%  This function plots the uv coordinates onto the image of
%  the calibration target.  It also projects the XYZ ...
    coordinates
%  back into image coordinates using the  calibration matrix
%  and plots these points too as a visual check on the ...
    accuracy of
%  the calibration process.
%  Lines from the origin to the vanishing points in the X, ...
    Y and
%  Z directions are overlaid on the image.
%  The mean squared error between the  positions of the uv ...
    coordinates
%  and the projected XYZ coordinates is also reported.
%
%  The function should also report the error in satisfying the
%  camera calibration matrix constraints.

```

```
29      %In this method, the main task is constract matrix A, ...
            then get p
30      [pointNum, ¬] = size(uv);
31
32      A = zeros(pointNum*2, 12);
33
34      for AIndex = 1 : pointNum
35
36          X = XYZ(AIndex, 1);
37          Y = XYZ(AIndex, 2);
38          Z = XYZ(AIndex, 3);
39
40          u = uv(AIndex, 1);
41          v = uv(AIndex, 2);
42
43          A(AIndex*2 - 1, :) = [X, Y, Z, 1, 0, 0, 0, 0, ...
                -u*X, -u*Y, -u*Z, -u];
44          A(AIndex*2, :) = [0, 0, 0, 0, X, Y, Z, 1, -v*X, ...
                -v*Y, -v*Z, -v];
45
46      end
47
48
49      [¬, ¬, V] = svd(A,0);
50      P = reshape(V(:,end), 4, 3)';
51
52      % then, write a method to get reprojection error, get ...
            our transformed
53      % points
54      XYZone = [XYZ, ones(pointNum, 1)]'; % construct XYZone ...
            with four columns
55      uvTrans = P*XYZone; % get transformed uv values
56      uvTrans(1,:) = uvTrans(1,:)./uvTrans(3,:); % make ...
            transformed uv third row is 1
57      uvTrans(2,:) = uvTrans(2,:)./uvTrans(3,:);
58
59      % get Euclidean distance between re-projected points ...
            and original points
60      reprojectError = sqrt(sum(sum((uvTrans(1:2,:) - ...
            uv').^2)));
61
62      fprintf('The mean reprojection error is: %d.\n', ...
            reprojectError/pointNum);
63
```

```
64      % also show our select points in image, with ...
           coordinate calcuated by P
65      uvTrans = uvTrans';
66
67      figure('Name', 'Selection Points Compare with ...
           Calcuated Points');
68      imshow(im);
69      hold on;
70      plot(uv(:,1),uv(:,2),'g+');
71      plot(uvTrans(:,1),uvTrans(:,2),'ro');
72      hold off;
73
74  end
```

```
1   %VGG_KR_FROM_P Extract K, R from camera matrix.
2   %
3   %     [K,R,t] = VGG_KR_FROM_P(P [,noscale]) finds K, R, t ...
        such that P = K*R*[eye(3) -t].
4   %     It is det(R)==1.
5   %     K is scaled so that K(3,3)==1 and K(1,1)>0. Optional ...
        parameter noscale prevents this.
6   %
7   %     Works also generally for any P of size N-by-(N+1).
8   %     Works also for P of size N-by-N, then t is not computed.
9
10
11  % Author: Andrew Fitzgibbon <awf@robots.ox.ac.uk>
12  % Modified by werner.
13  % Date: 15 May 98
14
15
16  function [K, R, t] = vgg_KR_from_P(P, noscale)
17
18  N = size(P,1);
19  H = P(:,1:N);
20
21  [K,R] = vgg_rq(H);
22
23  if nargin < 2
24    K = K / K(N,N);
25    if K(1,1) < 0
26      D = diag([-1 -1 ones(1,N-2)]);
27      K = K * D;
28      R = D * R;
```

```
29
30    %  test = K*R;
31    %  vgg_assert0(test/test(1,1) - H/H(1,1), 1e-07)
32    end
33  end
34
35  if nargout > 2
36    t = -P(:,1:N)\P(:,end);
37  end
38
39  return
```

```
1  function [rotation_out] = comp_decomp_matrix(rotations)
2  %ROTO_MATRIX: calculate rotation matrix from euler angles ...
       in degrees
3  %    Input:  1x3 vector of euler rotations around x ...
       rotations(1), y
4  %    rotations(2), and z, rotations(3) or 3x3 rotation matrix
5  %    Output: 3x3 matrix representing rotations around x, y, ...
       and z axis
6  %            or 1x3 vector of euler angles in degrees
7  %    NOTE: Euler angles returned when doing a decomposition ...
       will be in the
8  %    following ranges (in radians):
9
10 %    theta_x --> (-pi, pi)
11 %    theta_y --> (-pi/2, pi/2)
12 %    theta_z --> (-pi, pi)
13
14 % Angles within these ranges will be the same after ...
       decomposition: angles
15 % outside these ranges will produce the correct rotation ...
       matrix, but the
16 % decomposed values will be different to the input angles.
17
18 % error check
19 if  size(rotations, 2) == 2 | size(rotations, 2) > 3
20     error('Input must be 1x3 vector of xyz euler rotations ...
           or 3x3 rotation matrix');
21 end
22
23 if  size(rotations, 1) == 2 | size(rotations, 2) > 3
24     error('Input must be 1x3 vector of xyz euler rotations ...
           or 3x3 rotation matrix');
```

```
25 end
26
27 if size(rotations, 1) == 1
28 % calculate rotation matrix
29 x_rot = [1 0 0; 0 cosd(rotations(1)) -sind(rotations(1)); ...
      0 sind(rotations(1)) cosd(rotations(1))];
30 y_rot = [cosd(rotations(2)) 0 sind(rotations(2)); 0 1 0; ...
      -sind(rotations(2)) 0 cosd(rotations(2))];
31 z_rot = [cosd(rotations(3)) -sind(rotations(3)) 0 ; ...
      sind(rotations(3)) cosd(rotations(3)) 0; 0 0 1];
32 rotation_out = x_rot*y_rot*z_rot;
33 else
34 rotation_out = zeros(1,3);
35 rotation_out(1) = atan2(rotations(3,2),rotations(3,3));
36 rotation_out(2) = atan2(-rotations(3,1), ...
      (sqrt((rotations(3,2)^2)+(rotations(3,3)^2))));
37 rotation_out(3) = atan2(rotations(2,1),rotations(1,1));
38 % convert back to degrees
39 for i=1:size(rotation_out,2)
40     rotation_out(i) = 180*rotation_out(i)/pi;
41 end
42 end
```