Table of Contents

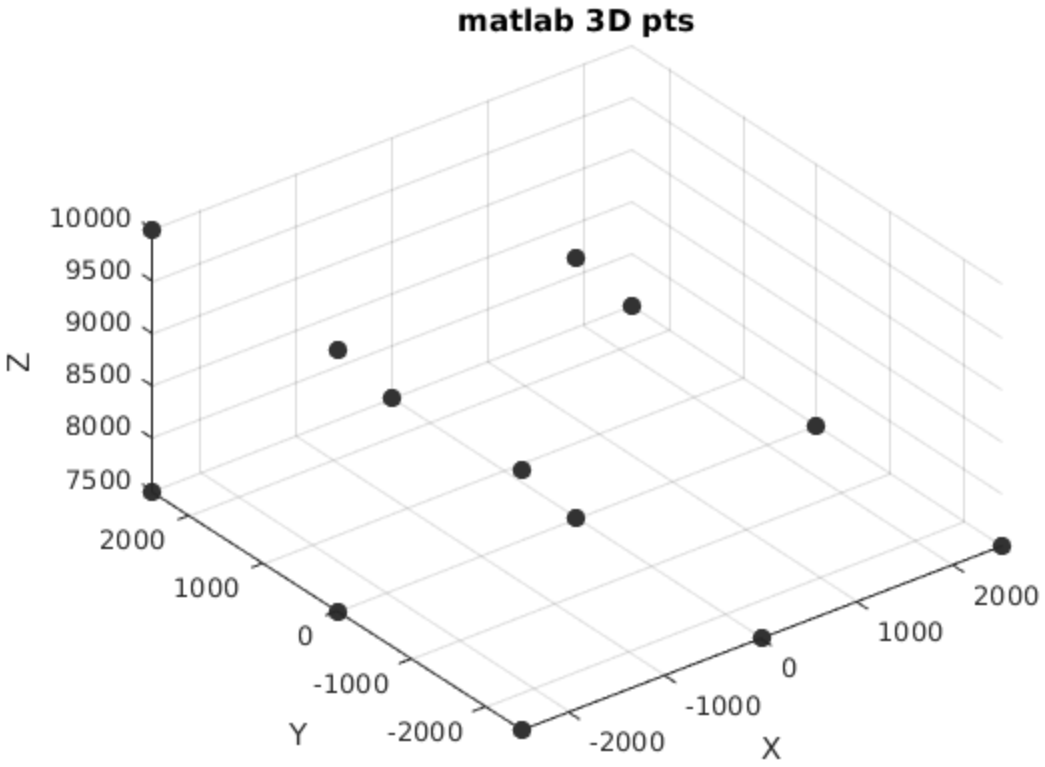
..... 1
Initialize model 1
Test the full reconstruction 4
Make the two results coincide: Three view geometry, or SFM 6


```
% TESTFILE - Tests the reconstruction algorithms on a simple model
 with
%            three cameras.
%
% Syntax:  TestFile
%
% Inputs:
%    None:  TestFile.m automaticly computes all necessary values from
 a
%           simple model.
%
% Outputs:
%    Vizualizations:
%                        - 3D original points of simple model
%                        - 2D points of projections on three cameras
%                        - 3D reconstructed points
%
% Other m-files required: - all from ./test folder
%                           - cv_triangulate
%                           - compute_fund_mat
% Subfunctions: none
% MAT-files required: none

% Author:   Thomas Lew
% email:    thomas.lew@epfl.ch
% Website:  https://github.com/thomasjlew/
% April 2017; Last revision: 2-May-2017

%------------- BEGIN CODE --------------

close all;
clear all;

addpath('./test')
```

Initialize model

```
% Intrinsics
K  = [-2500,    0, 1500;
          0,-2500, 1000;
          0,    0,    1];
% First Camera
R1 = eye(3);
C1 = zeros(3,1);
```

```
Rt1 = [R1', -R1'*C1];
P1 = K * Rt1;
% Second camera
R2 = my_rotationVectorToMatrix( [0;1;0], pi/4 );
C2 = [-10000*sqrt(2)/2;0;10000*(1-sqrt(2)/2)]; % see http://
ksimek.github.io/2012/08/22/extrinsic/
Rt2 = [R2', -R2'*C2];  % RT2 describes how the world is transformed
P2 = K * Rt2;          % relative to the camera.
% 3rd camera
R3 = my_rotationVectorToMatrix( [0;1;0], pi/4 );
C3 = [-10000;0;10000];
Rt3 = [R3', -R3'*C3];
P3 = K * Rt3;

% Points
% 1st Camera ( also Global frame )
homog_3d_pts = init3dpts;
visualize_3d_pts(homog_3d_pts, 'matlab 3D pts')

% Project the points onto the two camera image planes
[proj_pts1, proj_pts2, proj_pts3] = project_pts(P1, P2, P3,
 homog_3d_pts);

% View the points as an image for each camera
plot_2d_pts(proj_pts1,proj_pts2, proj_pts3);

% Test the triangulation function (WORKS)
% [ X ] = cv_triangulate(proj_pts1, proj_pts2, P1, P2);
% visualize_3d_pts(X, 'reconstructed pts');
```


matlab 3D pts
Z
10000
9500
9000
8500
8000
7500
Y
2000
1000
0
-1000
-2000
X
-2000
-1000
0
1000
2000

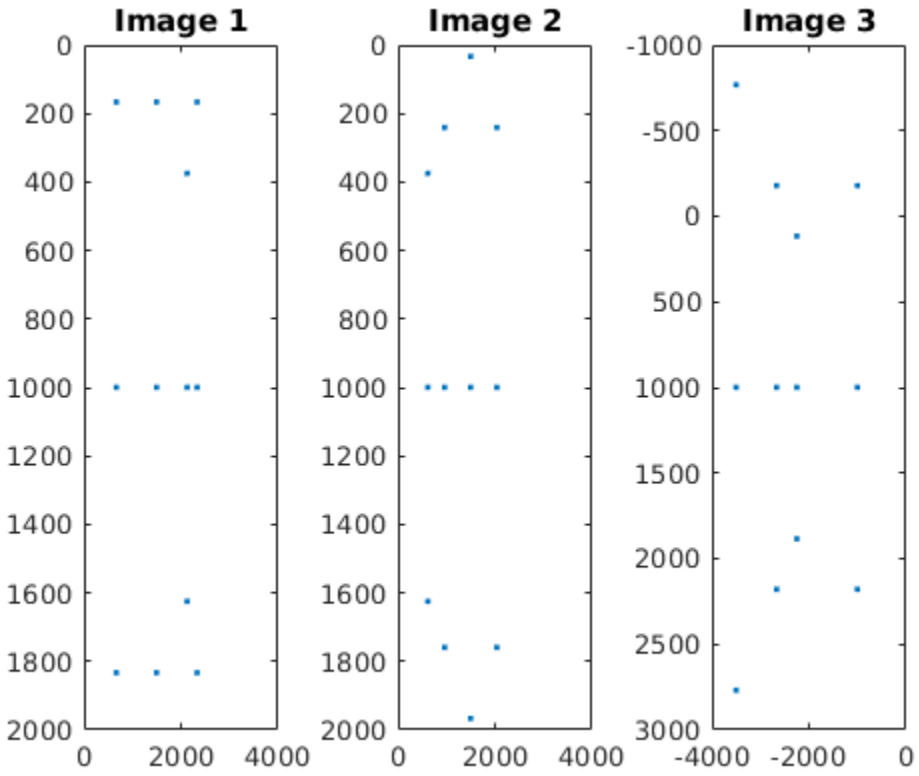


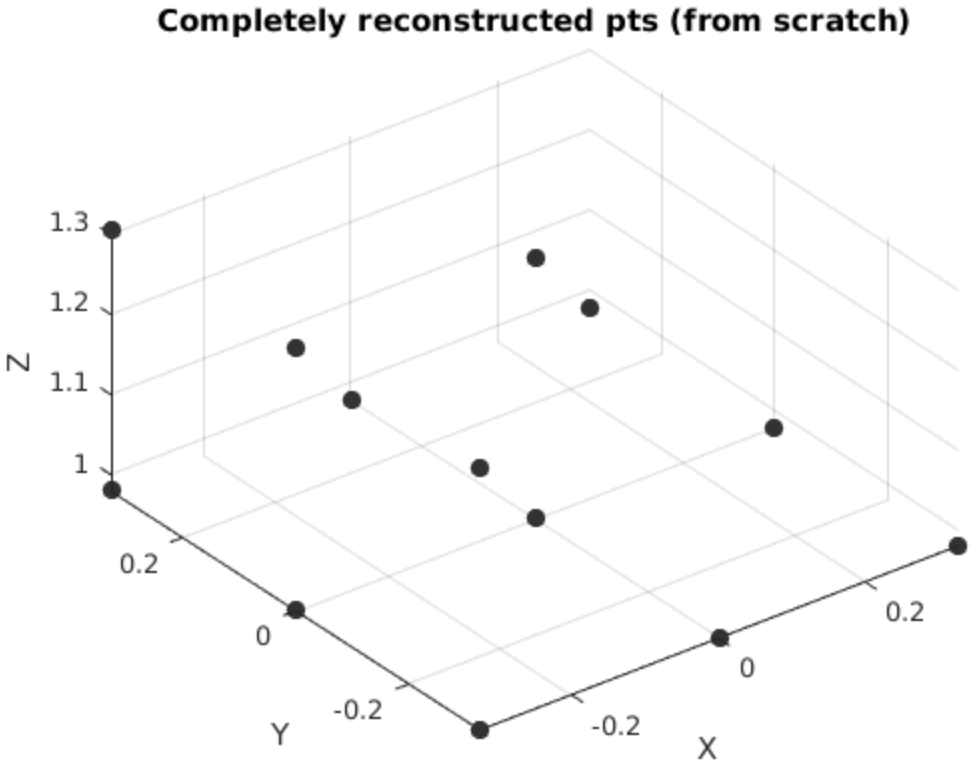
Image 1
Image 2
Image 3
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
2000
4000
-1000
-500
0
500
1000
1500
2000
2500
3000
-4000
-2000
0

Test the full reconstruction

Recover 2d points as seen by three cameras

```
v1 = proj_pts1(:,1:2);
v2 = proj_pts2(:,1:2);
v3 = proj_pts3(:,1:2);

%   Intrinsics
% K = [-2759.48,     0, 1520.69;
%      0,    -2764.16, 1006.81;
%      0,           0,    1]

%   Fundamental matrix
% save_fundMat !!!!!!!!!! this function doesnt work with some formats
[ F12, err_F12] = compute_fund_mat( v1, v2 );
[ F23, err_F23] = compute_fund_mat( v2, v3 );

% Computation of the Essential matrix
E12 = K' * F12 * K;
E23 = K' * F23 * K;

% Computation of camera matrix from essential matrix and intrinsics
pt1 = v1(1,:); pt2 = v2(1,:);   % Best match to check [R|t] solution
[R12, t12] = get_Rt_from_essential_mat(E12, K, pt1, pt2);
pt2 = v2(1,:); pt3 = v3(1,:);   % Best match to check [R|t] solution
[R23, t23] = get_Rt_from_essential_mat(E23, K, pt2, pt3);

% Form the camera matrices from intrinsics and extrinsics
P1 = K*[eye(3), zeros(3,1)];
P2 = K*[R12, t12];
P3 = K*[R23, t23];

%   Triangulate
X12 = cv_triangulate(v1, v2, P1, P2);
X23 = cv_triangulate(v2, v3, P1, P3);   % P2 == eye|0 since it's now
 origin

%   Show the results
visualize_3d_pts(X12, 'Completely reconstructed pts (from scratch)');
visualize_3d_pts(X23, 'image pair 23');
```

```
Warning: Recover [R|t]: Solution 4
Warning: Recover [R|t]: Solution 1
```


Completely reconstructed pts (from scratch)
1.3
1.2
1.1
1
Z
0.2
0
-0.2
Y
-0.2
0
0.2
X

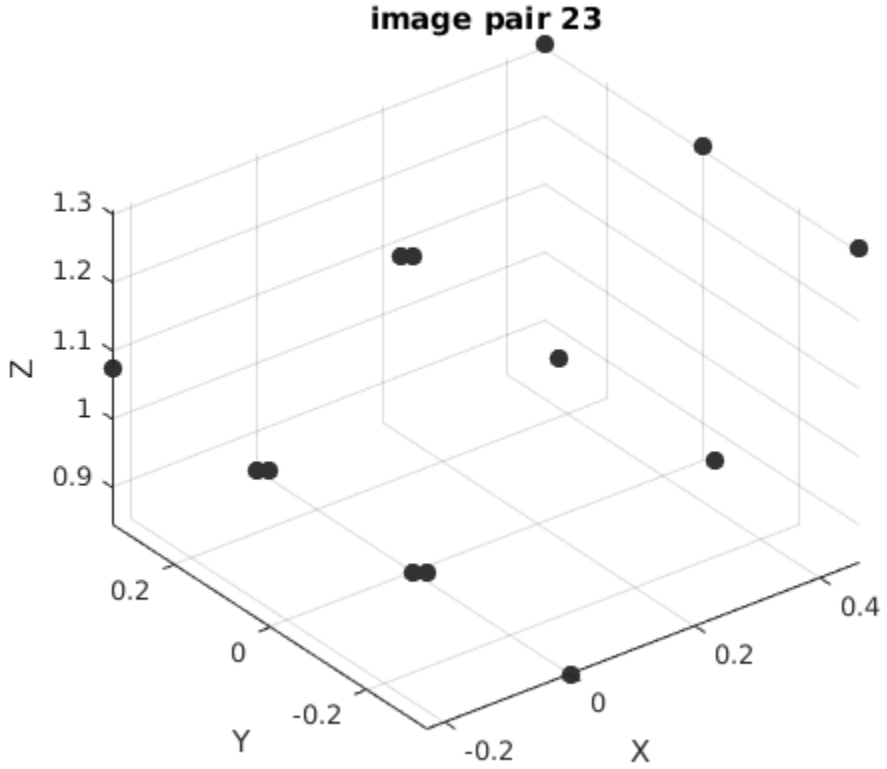


image pair 23
1.3
1.2
1.1
1
0.9
Z
0.2
0
-0.2
Y
-0.2
0
0.2
0.4
X

Make the two results coincide: Three view geometry, or SFM

```
Adj_01_inv = [Rt2; 0 0 0 1];
Adj_12_inv = [Rt3; 0 0 0 1];
Adj_02_inv = Adj_12_inv * Adj_01_inv;
X_original1 = (Adj_01_inv * X23')';
X_original2 = (Adj_02_inv * X23')';

% An obtain rotation of 180° or pi[rad] is not physically correct
if(R12(1,1)<0)
    R12 = -R12;
end

% Recover 3D points and translation scales
alpha = 1;   %scalar for p (see pdf of final report)
beta  = 1;   %scalar for t
% Reconstructed point used for checking the 3d pts scale.
% Its x and y values need to be different from zero.
p = X23(6,1:3)';
A = R12 * homog_3d_pts(6,1:3)';
beta  = ((A(1)*p(2)/(p(1)*t12(2)))-A(2)/t12(2) ) / ...
                    (1- t12(1)*p(2)/(p(1)*t12(2)))
alpha = A(1)/p(1) + beta*t12(1)/p(1)

% Recover points in first frame
Adj_01_inv = [R12', -R12' * beta* t12; 0 0 0 1];
X23(:,1:3) = alpha * X23(:,1:3);
X_2_from_12 = (Adj_01_inv * X23')';
visualize_3d_pts(X_2_from_12, 'X2 from 23 with Resize');

%------------- END OF CODE --------------
```

```
beta =

  -7.6540e+03


alpha =

   7.6535e+03
```

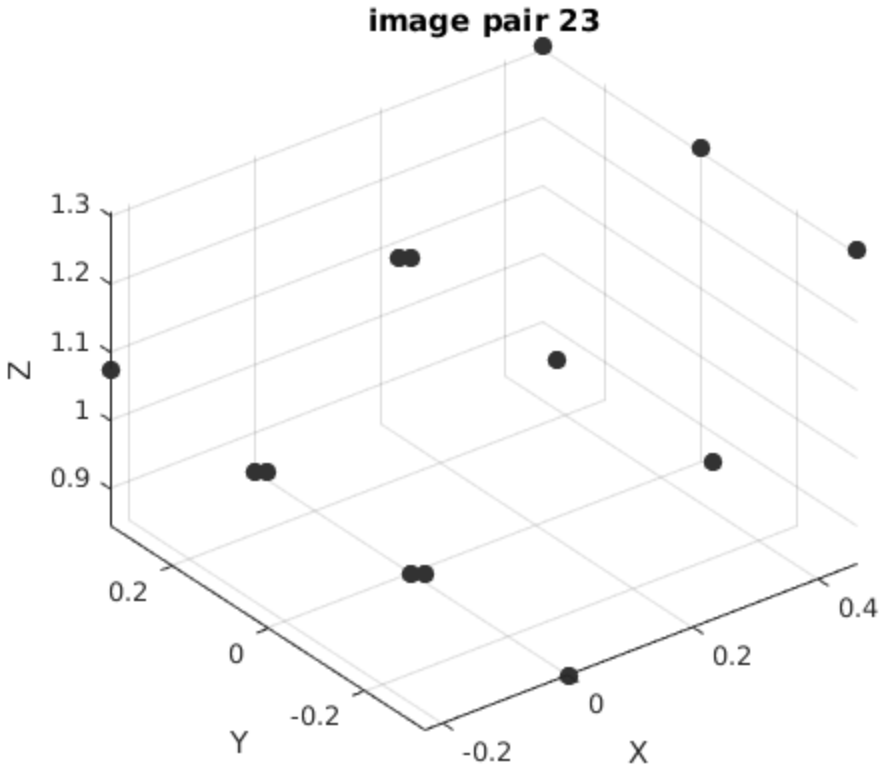
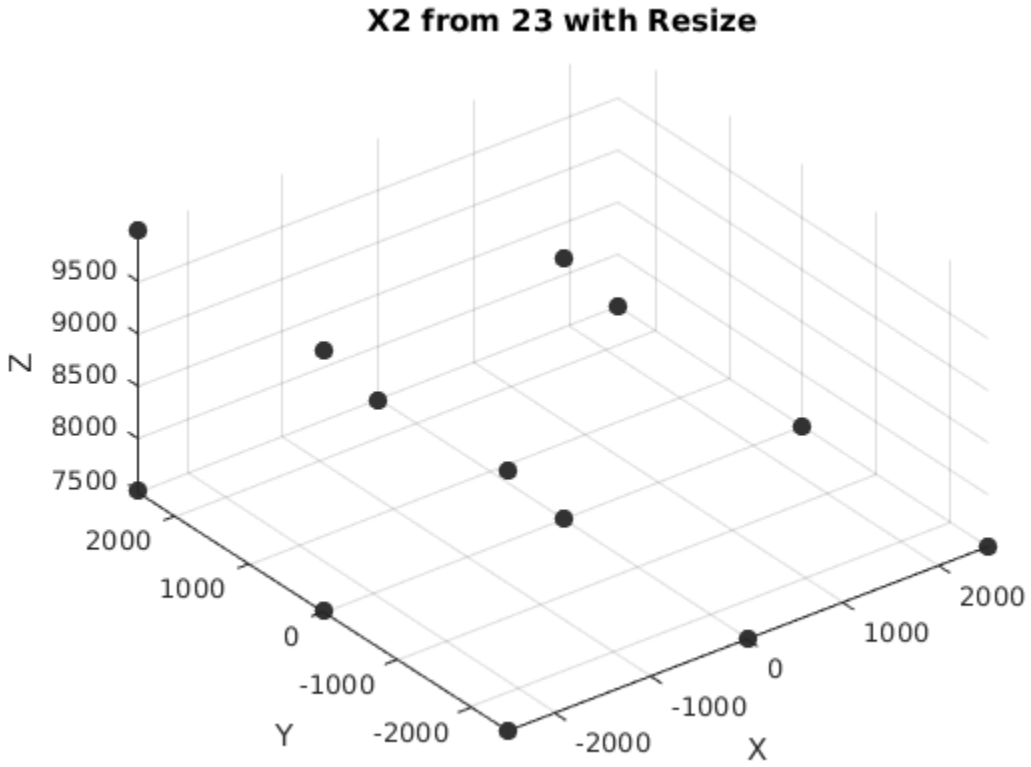


image pair 23
1.3
1.2
1.1
1
0.9
Z
0.2
0
-0.2
Y
-0.2
0
0.2
0.4
X


X2 from 23 with Resize
9500
9000
8500
8000
7500
Z
2000
1000
0
-1000
-2000
Y
-2000
-1000
0
1000
2000
X

Published with MATLAB® R2016b