CLab3 Report

ENGN6528
You Li
u6430173

24 May 2019

Contents

Chapter 1

Task1 Two-View DLT based homography estimation.

1.1 Selected points

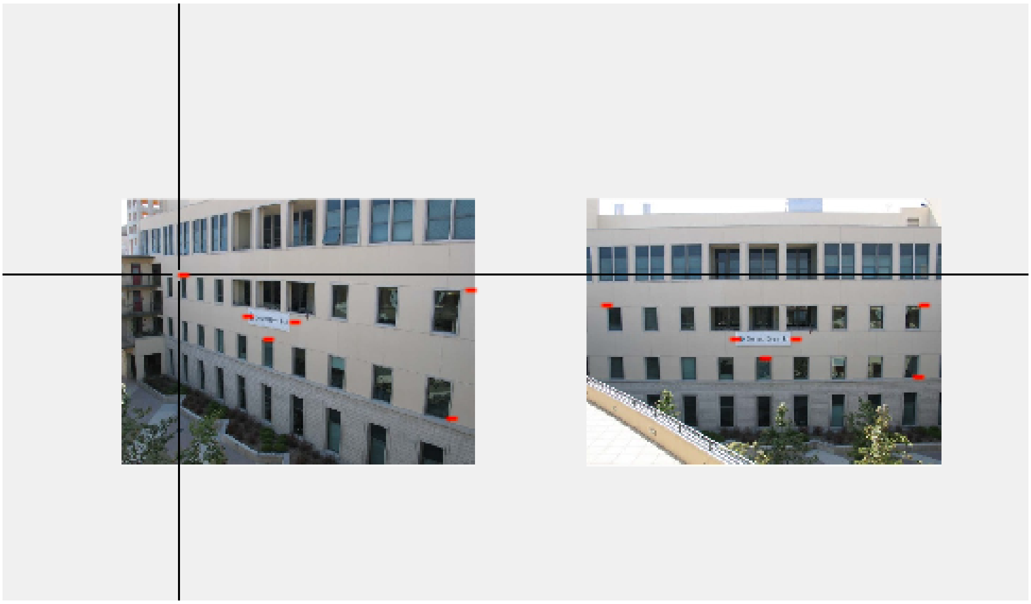

Figure 1.1: task1.4

1.2 Source Code of *Homography*

```matlab
function H = homography(u2Trans, v2Trans, uBase, vBase)
% Computes the homography H applying the Direct Linear Transformation
% The transformation is such that
% p = H p' , i.e.,:
% (uBase, vBase, 1)'=H*(u2Trans , v2Trans, 1)'
%
% INPUTS:
% u2Trans, v2Trans - vectors with coordinates u and v of the transformed image
% uBase, vBase - vectors with coordinates u and v of the original base image po
%
% OUTPUT
% H - a 3x3 Homography matrix
%
unit=ones(6,1);
% image coordinates in the Base image planar
p1=[uBase vBase unit]';
% image coordinates in the Trans image planar
p2=[u2Trans v2Trans unit]';

[p1_norm, p1_trans]= normalise2dpts(p1);
[p2_norm, p2_trans]= normalise2dpts(p2);


% to compute the matrix H, we should construct matrix A
A=[];
tempA=[];
% for every point, we have only two linearly independent equations
% I combine them together and get a 12x9 matrix A
for i=1:6
    tempA=[zeros(1,3) (-1)*p2_norm(:,i)' p1_norm(2,i)*p2_norm(:,i)';
        p2_norm(:,i)' zeros(1,3) -p1_norm(1,i)*p2_norm(:,i)'];
    A=[A ;tempA];
end

% conduct SVD on matrix A
[U,S,V]=svd(A);
% the right-most-column of V is the h with least squares
```

```
h=V(:,end);
H=reshape(h,[3, 3])';

% denormalise H, according to the transformation matrixs
H = inv(p1_trans)*H*(p2_trans);
end

% NORMALISE2DPTS - normalises 2D homogeneous points
%
% Function translates and normalises a set of 2D homogeneous points
% so that their centroid is at the origin and their mean distance from
% the origin is sqrt(2).  This process typically improves the
% conditioning of any equations used to solve homographies, fundamental
% matrices etc.
%
% Usage:   [newpts, T] = normalise2dpts(pts)
%
% Argument:
%   pts -  3xN array of 2D homogeneous coordinates
%
% Returns:
%   newpts -  3xN array of transformed 2D homogeneous coordinates.  The
%             scaling parameter is normalised to 1 unless the point is at
%             infinity.
%   T      -  The 3x3 transformation matrix, newpts = T*pts
%
% If there are some points at infinity the normalisation transform
% is calculated using just the finite points.  Being a scaling and
% translating transform this will not affect the points at infinity.

function [newpts, T] = normalise2dpts(pts)

    if size(pts,1) ~= 3
        error('pts must be 3xN');
    end

    % Find the indices of the points that are not at infinity
    finiteind = find(abs(pts(3,:)) > eps);

```

```
    if length(finiteind) ~= size(pts,2)
        warning('Some points are at infinity');
    end

    % For the finite points ensure homogeneous coords have scale of 1
    pts(1,finiteind) = pts(1,finiteind)./pts(3,finiteind);
    pts(2,finiteind) = pts(2,finiteind)./pts(3,finiteind);
    pts(3,finiteind) = 1;

    c = mean(pts(1:2,finiteind)')';            % Centroid of finite points
    newp(1,finiteind) = pts(1,finiteind)-c(1); % Shift origin to centroid.
    newp(2,finiteind) = pts(2,finiteind)-c(2);

    dist = sqrt(newp(1,finiteind).^2 + newp(2,finiteind).^2);
    meandist = mean(dist(:));  % Ensure dist is a column vector for Octave 3.0.

    scale = sqrt(2)/meandist;

    T = [scale   0   -scale*c(1)
         0     scale -scale*c(2)
         0       0      1      ];

    newpts = T*pts;
end

function im_warped = ImageWarping(im, H)
        [u_x, u_y] = GetPointsToTransform(size(im,2), size(im,1));
        [v_x, v_y] = TransformPointsUsingHomography(inv(H), u_x, u_y);
        im_warped = BuildWarpedImage(double(im), v_x, v_y);
        im_warped = uint8(im_warped);
end

function [u_x, u_y] = GetPointsToTransform(width, height)
        [u_x, u_y] = meshgrid(1:width, 1:height);
end

function [v_x, v_y] = TransformPointsUsingHomography(H, u_x, u_y)
        v_x = H(1,1)*u_x + H(1,2)*u_y + H(1,3);
        v_y = H(2,1)*u_x + H(2,2)*u_y + H(2,3);
```

```
        v_z = H(3,1)*u_x + H(3,2)*u_y + H(3,3);

        v_x = v_x./v_z;
        v_y = v_y./v_z;
end

function im_warped = BuildWarpedImage(im, v_x, v_y)
        h = size(v_x, 1); w = size(v_x,2);
        im_warped(:,:,1) = reshape(interp2(im(:,:,1), v_x(:), v_y(:)), [h, w]);
        im_warped(:,:,2) = reshape(interp2(im(:,:,2), v_x(:), v_y(:)), [h, w]);
        im_warped(:,:,3) = reshape(interp2(im(:,:,3), v_x(:), v_y(:)), [h, w]);
end
```

end of matlab code.

1.3 $uvpointsand3 \times 3$ H matrix

```
uv =

   59.6979  113.1471
  475.3984  131.5642
  172.8316  165.7674
  235.9759  173.6604
  199.1417  202.6016
  456.9813  315.7353
   30.5957  152.7872
  467.6383  152.7872
  205.9362  197.2766
  284.4468  194.6596
  239.9574  220.8298
  467.6383  249.6170


H =

   -0.2992   -0.1447  -14.6125
    0.0745   -0.8213   41.9147
    0.0009   -0.0004   -0.7033
```

1.4 Homography Result

Figure 1.2: task1.4

1.5 Factors of Homography Results

1.5.1 *Pointnumbers*

In the case where there are more than 4 point correspondences available, I took 12 points to provide more references for calculating the H matrix. And this minimizes the error generated from points. Some experiments were tried and the factors were listed.

1.5.2 *Distanceofselectedpoints*

The simplest way to minimize the error is to minimize the algebraic distance of selected points.To ensure that the h selected isn't the zero vector we add the constraint that $||h|| = 1$. The unit singular vector corresponding to the smallest singular value of A. This can be found using SVD. However, this is not so geometrically meaningful.

1.5.3 *Geometricdistance*

The geometric distance measures the Euclidian image distance between where the homography maps a point and where the point's correspondence was originally found. This definces the transfer error.

The correspondences $\mathbf{x_i} \rightarrow \mathbf{x_i'}$ is:

$$\sum_i d\left(\mathbf{x_i'}, H\mathbf{x_i}\right)^2$$

Where d is the gepmetry distance. With consider of forward and backward transpose,

$$\sum_i d\left(\mathbf{x_i'}, H\mathbf{x_i}\right)^2 + d\left(\mathbf{x_i}, H^{-1}\mathbf{x_i'}\right)^2$$

Chapter 2

3D-2D Camera Calibration

2.1 Implementation of Calibrate

```
%% Task 2
img_a = imread('CLab3_Materiala/stereo2012a.jpg');
XYZ = load(CLab3_Materiala/XYZ.txt)
[u, v] = load(CLab3_Materiala/ux.txt)
plot(u, v, 'ro');

C = calibrate(img_a,XYZ,[u,v]);
disp(C);
%% calc K, R, T
[K, R, t] = vgg_KR_from_P(C);
disp('K = ');
disp(K);
disp('R = ');
disp(R);
disp('t = ');
disp(t);
%calculate focal
alpha = K(1,1);
gama = K(1,2);
thelta = acot(gama/-alpha);
fx = abs(alpha);
fy = abs(K(2,2)*sin(thelta));
f = sqrt(fx^2+fy^2);
```

```
disp(['fx: ',num2str(fx)]);
disp(['fy: ',num2str(fy)]);
disp(['focal: ',num2str(f)]);

%% calibrate function
function C = calibrate(im, XYZ, uv)
[len,~] = size(uv);

A = zeros(len*2,12);

for tmp = 1:len
   %get the 3D vertices and their 2D corresponding points
   X = XYZ(tmp,1);
   Y = XYZ(tmp,2);
   Z = XYZ(tmp,3);

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

   A(tmp*2-1,:) = [X,Y,Z,1,0,0,0,0,-u*X,-u*Y,-u*Z,-u];
   A(tmp*2,:) = [0,0,0,0,X,Y,Z,1,-v*X,-v*Y,-v*Z,-v];
end
[~,~,V] = svd(A);

C = V(:,end);
C = C/C(end);
C = reshape(C,[4 3]);
C = C';

imshow(im);
hold on;
plot(uv(:,1),uv(:,2),'ro');
hold on;
xyz1 = [XYZ,ones(tmp,1)]';
xy = C*xyz1;

xy(1,:) = xy(1,:)./xy(3,:);
xy(2,:) = xy(2,:)./xy(3,:);

```

```
distance = (xy(1:2,:) - uv').^2;
distance = mean(sum(distance));
disp(['Mean Square Error: ',num2str(distance)]);

plot(xy(1,:),xy(2,:),'g+');
end
```

2.2 Experiment Result of 2012a

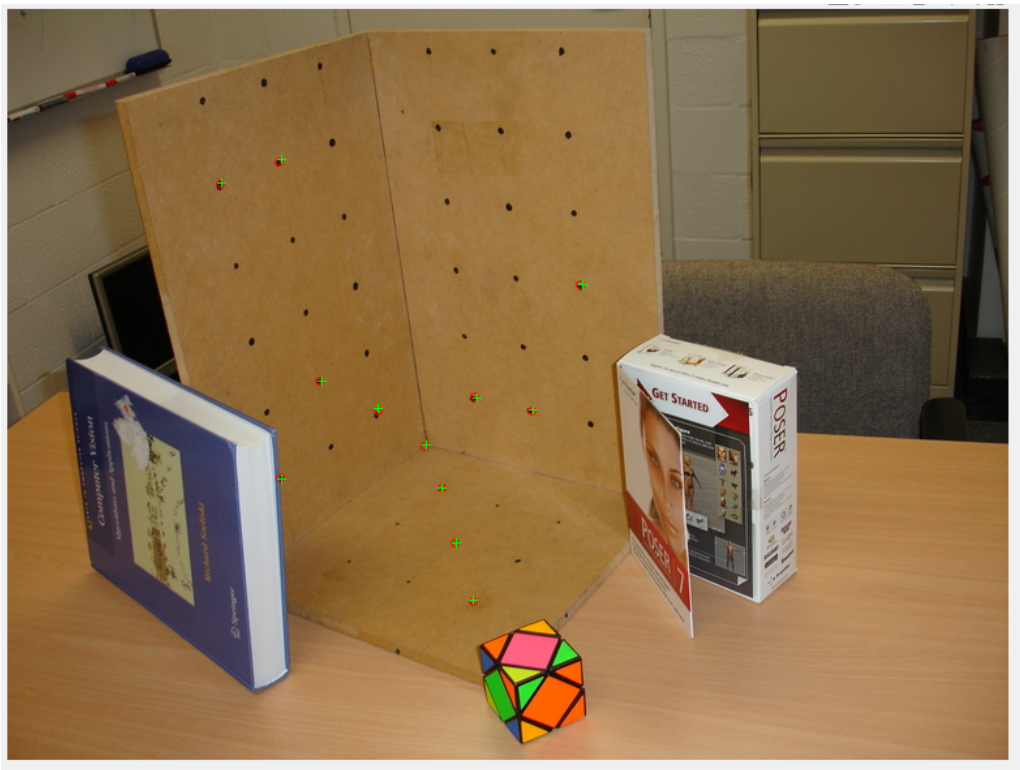

Figure 2.1: task2

2.3 3×4 Matrix C

```
C =
-15.5043  -10.7309   -6.9259  315.9230
-16.4116  -10.9395   -7.3210  325.9015
-0.0475   -0.0323   -0.0234    1.0000

MSE =
2.7535
```

2.4 Matrix K, R, t

Decompose the C matrix into K, R, t, such that C = K[R|t]

```
K =
    5.4072    9.9570  323.7067
         0   10.9062  339.0619
         0         0    1.0000

R =
    0.4505   -0.8528    0.2642
   -0.4590    0.0326    0.8878
   -0.7657   -0.5213   -0.3767

t =
   -2.2690
   19.8003
   20.0155
```

2.5 Focal length and Angle

2.5.1 what is the focal length of the camera, and in what unit of measurement?

```
fx: 5.4072
fy: 5.2047
focal: 7.5051
```

The focal length of the camera is 7.5 cm.

2.5.2 what is the pitch angle of the camera with respect to the X-Z plane in the world coordinate system?

```
theta =
88.995
```

The angle of the camera should be around 90 degree to X-Z plane.