Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science
http://inside.mines.edu/~whoff/
Linear Pose Estimation Examples
DLT for Pose Estimation

• Run DLT pose estimation code on “img1_rect.tif”
  – How does the result compare to the nonlinear pose estimation algorithm?
% Pose estimation using DLT algorithm (direct linear transform)
clear all
close all

I = imread('img1_rect.tif');
imshow(I, [])

% These are the points in the model's coordinate system (inches)
P_M = [  0   0   2   0   0   2;
       10   2   0  10   2   0;
        6   6   6   2   2   2;
         1   1   1   1   1   1 ];
N = size(P_M,2);

% Define camera parameters
f = 715;      % focal length in pixels
cx = 354;
cy = 245;

K = [  f  0  cx;  0  f  cy;  0  0  1 ];  % intrinsic parameter matrix

% Here are the observed image points (a 3xN matrix)
p = [  
   183 147  1;     % 1
   350 133  1;     % 2
   454 144  1;     % 3
   176 258  1;     % 4
   339 275  1;     % 5
   444 286  1];    % 6
p = p';

hold on
plot(p(1,:), p(2,:), 'g*');
% Solve for the pose of the model with respect to the camera.

pn = inv(K)*p; % Normalize image points

% Ok, now we have pn = Mext*P_M.
% If we know P_M and pn, we can solve for the elements of Mext.
% The equations for x,y are:
%   x = (r11*X + r12*Y + r13*Z + tx)/(r31*X + r32*Y + r33*Z + tz)
%   y = (r21*X + r22*Y + r23*Z + ty)/(r31*X + r32*Y + r33*Z + tz)
% or
%   r11*X + r12*Y + r13*Z + tx - x*r31*X - x*r32*Y - x*r33*Z - x*tz = 0
%   r21*X + r22*Y + r23*Z + ty - y*r31*X - y*r32*Y - y*r33*Z - y*tz = 0

% Put elements of Mext into vector w:
%   w = [r11 r12 r13 r21 r22 r23 r31 r32 r33 tx ty tz]
% We then have Ax = 0. The rows of A are:
%   X  Y  Z  0  0  0   -x*X   -x*Y   -x*Z   1   0   -x
%   0  0  0   X  Y  Z   -y*X   -y*Y   -y*Z   1   0   -y

A = zeros(N,12);
for i=1:N
    X = P_M(1,i);   Y = P_M(2,i);   Z = P_M(3,i);
    x = pn(1,i);    y = pn(2,i);
    A( 2*(i-1)+1, :) = [ X  Y  Z  0  0  0   -x*X   -x*Y   -x*Z   1   0   -x ];
    A( 2*(i-1)+2, :) = [ 0  0  0   X  Y  Z   -y*X   -y*Y   -y*Z   0   1   -y ];
end

% Solve for the value of x that satisfies Ax = 0.
% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,end); % get last column of V
% Reshape x back to a 3x4 matrix, M = [R_m_c  tmorg_c]
M = [ x(1)  x(2)  x(3)  x(10);
     x(4)  x(5)  x(6)  x(11);
     x(7)  x(8)  x(9)  x(12) ];

% We can find the camera center, tcorg_m by solving the equation MX=0.
% To see this, write M = [R_m_c  tmorg_c]. But tmorg_c = -R_m_c * tcorg_m.
% So M = R_m_c*[ I  -tcorg_m ]. And if we multiply M times tcorg_m, we
% get   R_m_c*[ I  -tcorg_m ] * [tcorg_m; 1] = 0.
[U,D,V] = svd(M);
tcorg_m = V(:,end);  % Get last column of V
tcorg_m = tcorg_m / tcorg_m(4);  % Divide through by last element

% Get rotation portion from M
[Q,B] = qr(M(1:3,1:3)');

% Enforce that the diagonal values of B are positive
for i=1:3
    if B(i,i)<0
        B(i,:) = -B(i,:);  % Change sign of row
        Q(:,i) = -Q(:,i);  % Change sign of column
    end
end

Restimated_m_c = Q';  % Estimated rotation matrix, model-to-camera

% R must be a right handed rotation matrix; ie det(R)>0
if det(Restimated_m_c)<0
    Restimated_m_c = -Restimated_m_c;
end
% Final estimated pose
Restimated_c_m = Restimated_m_c';
Hestimated_c_m = [Restimated_c_m tcorg_m(1:3); 0 0 0 1];
disp('Final computed pose, H_c_m:'), disp(Hestimated_c_m);

Hestimated_m_c = inv(Hestimated_c_m);
disp('Final computed pose, H_m_c:'), disp(Hestimated_m_c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reproject points back onto the image
M = Hestimated_m_c(1:3,:);
p = K*M*P_M;
p(1,:) = p(1,:)./p(3,:);
p(2,:) = p(2,:)./p(3,:);
p(3,:) = p(3,:)./p(3,:);
plot(p(1,:), p(2,:), 'r*');
Recall that the image point projection of a target marker is
\[ x = \frac{r_{11}X + r_{12}Y + r_{13}Z + t_x}{r_{31}X + r_{32}Y + r_{33}Z + t_z}, \quad y = \frac{r_{21}X + r_{22}Y + r_{23}Z + t_y}{r_{31}X + r_{32}Y + r_{33}Z + t_z} \]

or
\[
\begin{align*}
    x(r_{31}X + r_{32}Y + r_{33}Z + t_z) &= r_{11}X + r_{12}Y + r_{13}Z + t_x \\
y(r_{31}X + r_{32}Y + r_{33}Z + t_z) &= r_{21}X + r_{22}Y + r_{23}Z + t_y
\end{align*}
\]

Now, the (X,Y,Z) of the marker point is unknown and everything else is known.

Rearrange to put into the form \( A \mathbf{x} = \mathbf{b} \) where \( \mathbf{x} = (X,Y,Z) \) and solve for \( \mathbf{x} \).

Note that we will need multiple cameras (how many?)

Note that each camera has its own parameters \( (r_{11}, r_{12}, \ldots, t_y, t_z) \).
DTT for Reconstruction

- We have
  \[ x(r_3X + r_2Y + r_3Z + t_z) = r_{11}X + r_{12}Y + r_{13}Z + t_x \]
  \[ y(r_3X + r_2Y + r_3Z + t_z) = r_{21}X + r_{22}Y + r_{23}Z + t_y \]

- The unknown is \( \mathbf{x} = (X,Y,Z) \); ie, the point in the world
- Put into the form \( \mathbf{A} \mathbf{x} = \mathbf{b} \)

\[
\begin{pmatrix}
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\end{pmatrix}
\begin{pmatrix}
wX \\
wY \\
wZ \\
\end{pmatrix}
= 
\begin{pmatrix}
\_ \\
\_ \\
\_ \\
\end{pmatrix}
\]

How many measurements of the point (ie, number of cameras) do we need to solve for \( \mathbf{x} \)?
DTL for Reconstruction

- We have
  
  \[ \begin{align*}
  r_{11}X + r_{12}Y + r_{13}Z + t_x - x(r_{31}X + r_{32}Y + r_{33}Z + t_z) &= 0 \\
  r_{21}X + r_{22}Y + r_{23}Z + t_y - y(r_{31}X + r_{32}Y + r_{33}Z + t_z) &= 0
  \end{align*} \]

- The unknown is \( x = (X,Y,Z) \); i.e., the point in the world
- Put into the form \( Ax = b \)
  
  \[
  \begin{pmatrix}
  r_{11} - xr_{31} & r_{12} - x & r_{32} & r_{13} - x & r_{33} \\
  r_{21} - yr_{31} & r_{22} - y & r_{32} & r_{23} - y & r_{33}
  \end{pmatrix}
  \begin{pmatrix}
  X \\
  Y \\
  Z
  \end{pmatrix}
  =
  \begin{pmatrix}
  -t_x + xt_z \\
  -t_y + yt_z
  \end{pmatrix}
  \]

- How many measurements of the point (i.e., number of cameras) do we need to solve for \( x \)?
DLT for Reconstruction

• Note that each camera has its own parameters \( (r_{11}, r_{12}, ..., t_y, t_z) \)
• So for example, with two cameras

\[
\begin{pmatrix}
\text{________} & \text{________} & \text{________} \\
\text{________} & \text{________} & \text{________} \\
\text{________} & \text{________} & \text{________} \\
\text{________} & \text{________} & \text{________}
\end{pmatrix}
\begin{pmatrix}
wX \\
wY \\
wZ
\end{pmatrix} = \begin{pmatrix}
\text{________} \\
\text{________} \\
\text{________}
\end{pmatrix}
\]
DLT for Reconstruction

• Note that each camera has its own parameters \( (r_{11}, r_{12}, \ldots, t_x, t_z) \)

• So for example, with two cameras

\[
\begin{pmatrix}
  r_{11} - x r_{31} & r_{12} - x r_{32} & r_{13} - x r_{33} \\
  r_{21} - y r_{31} & r_{22} - y r_{32} & r_{23} - y r_{33}
\end{pmatrix}
\begin{pmatrix}
  X \\
  Y \\
  Z
\end{pmatrix} = \begin{pmatrix}
  -t_x + xt_z \\
  -t_y + yt_z
\end{pmatrix}
\]

\[
\begin{pmatrix}
  r_{11}^{(c1)} - x^{(c1)} r_{31}^{(c1)} \\
  r_{11}^{(c1)} - x^{(c1)} r_{31}^{(c1)}
\end{pmatrix}
\begin{pmatrix}
  r_{12}^{(c1)} - x^{(c1)} r_{32}^{(c1)} \\
  r_{12}^{(c1)} - x^{(c1)} r_{32}^{(c1)}
\end{pmatrix}
\begin{pmatrix}
  r_{13}^{(c1)} - x^{(c1)} r_{33}^{(c1)} \\
  r_{13}^{(c1)} - x^{(c1)} r_{33}^{(c1)}
\end{pmatrix}
\begin{pmatrix}
  X \\
  Y \\
  Z
\end{pmatrix} = \begin{pmatrix}
  -t_x^{(c1)} + x^{(c1)} t_z^{(c1)} \\
  -t_y^{(c1)} + y^{(c1)} t_z^{(c1)}
\end{pmatrix}
\]

\[
\begin{pmatrix}
  r_{11}^{(c2)} - x^{(c2)} r_{31}^{(c2)} \\
  r_{11}^{(c2)} - x^{(c2)} r_{31}^{(c2)}
\end{pmatrix}
\begin{pmatrix}
  r_{12}^{(c2)} - x^{(c2)} r_{32}^{(c2)} \\
  r_{12}^{(c2)} - x^{(c2)} r_{32}^{(c2)}
\end{pmatrix}
\begin{pmatrix}
  r_{13}^{(c2)} - x^{(c2)} r_{33}^{(c2)} \\
  r_{13}^{(c2)} - x^{(c2)} r_{33}^{(c2)}
\end{pmatrix}
\begin{pmatrix}
  X \\
  Y \\
  Z
\end{pmatrix} = \begin{pmatrix}
  -t_x^{(c2)} + x^{(c2)} t_z^{(c2)} \\
  -t_y^{(c2)} + y^{(c2)} t_z^{(c2)}
\end{pmatrix}
\]

• where

\[ w^c H = \begin{pmatrix}
  r_{11}^{(c1)} & r_{12}^{(c1)} & r_{13}^{(c1)} & t_x^{(c1)} \\
  r_{21}^{(c1)} & r_{22}^{(c1)} & r_{23}^{(c1)} & t_y^{(c1)} \\
  r_{31}^{(c1)} & r_{32}^{(c1)} & r_{33}^{(c1)} & t_z^{(c1)} \\
  0 & 0 & 0 & 1
\end{pmatrix} \]

\[ \text{image point in camera 1} \begin{pmatrix}
  x^{(c1)} \\
  y^{(c1)}
\end{pmatrix} \]

(and similarly for camera 2)
Matlab code – reconstruction example

• We will make synthetic images of a scene, as if they were taken from two cameras

• First make some 3D points near the world origin

```matlab
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create some 3D points in the world
N = 10;
P_w = zeros(4,N);
for i=1:N
    r = 2 - i/N;
    z = 2*i/N;
    a = 4*pi*i/N;
    P_w(1,i) = r*cos(a);
P_w(2,i) = r*sin(a);
P_w(3,i) = z;
P_w(4,i) = 1;
end

plot3(P_w(1,:), P_w(2,:), P_w(3,:), 'o-');
axis equal
axis vis3d
```
Matlab code – reconstruction example

• Now place two cameras, looking at the origin
• We’ll just put the camera origins at:
  – Camera1: \((tx, ty, tz) = (-3, -6, 5)\)
  – Camera2: \((tx, ty, tz) = (+3, -6, 5)\)

• Since the camera looks at the origin, this determines the \(z\) axis of the camera
• It is just the unit vector from the \(c1\) origin to the world origin:

\[
^w \hat{z}_{c1} = \frac{-^w t_{c1org}}{\text{||}^w t_{c1org}\text{||}}
\]

• (similarly for camera 2)

\[
^w R_{c1} = \begin{pmatrix}
^w \hat{x}_{c1} & ^w \hat{y}_{c1} & ^w \hat{z}_{c1}
\end{pmatrix}
\]
Matlab code – reconstruction example

• To get the other columns of the rotation matrix:
• Assume that the x axis of the camera is in the x-y plane of the world
• Then the cross product of the camera’s z axis with the world z axis, points in the x axis of the camera

\[
\begin{align*}
{^w\hat{X}}_{c1} & = \frac{{^w\hat{Z}}_{c1} \times {^w\hat{Z}}_{w}}{|{^w\hat{Z}}_{c1} \times {^w\hat{Z}}_{w}|} \\
{^w\hat{Y}}_{c1} & = {^w\hat{Z}}_{c1} \times {^w\hat{X}}_{c1} \\
\end{align*}
\]

• The y axis of the camera can then be found as

• (similarly for camera 2)
Matlab code – reconstruction example

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Place two cameras in the world
% Place two cameras in the world
% Place two cameras in the world
t1org_w = [-3; -6; 5];  % Origin of camera 1 in world
uz = -t1org_w / norm(t1org_w);  % Camera z points toward origin
ux = cross(uz, [0;0;1]);    % Camera x is in the world xy plane
ux = ux/norm(ux);           % Camera x is in the world xy plane
uy = cross(uz, ux);         % Camera y is just z cross x
H_c1_w = [ ux uy uz t1org_w; 0 0 0 1];

% Origin of camera 2 in world
% Origin of camera 2 in world
% Origin of camera 2 in world
t2org_w = [+3; -6; 5];       % Origin of camera 2 in world
uz = -t2org_w / norm(t2org_w);  % Camera z points toward origin
ux = cross(uz, [0;0;1]);    % Camera x is in the world xy plane
ux = ux/norm(ux);           % Camera x is in the world xy plane
uy = cross(uz, ux);         % Camera y is just z cross x
H_c2_w = [ ux uy uz t2org_w; 0 0 0 1];

% Show cameras on the 3D plot
hold on
plot3(t1org_w(1), t1org_w(2), t1org_w(3), '*');
text(t1org_w(1)+0.1, t1org_w(2), t1org_w(3), '1');
plot3(t2org_w(1), t2org_w(2), t2org_w(3), '*');
text(t2org_w(1)+0.1, t2org_w(2), t2org_w(3), '2');

axis equal
axis vis3d
% Create intrinsic camera matrix
f = 400; % focal length in pixels
cx = 200;
cy = 200;
K = [ f 0 cx; 0 f cy; 0 0 1 ]; % intrinsic parameter matrix

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Project the points onto the two cameras

% Project points onto image1
H_w_c1 = inv(H_c1_w);
Mext = H_w_c1(1:3,:);
p1 = K*Mext*P_w;
p1(1,:) = p1(1,:)./p1(3,:);
p1(2,:) = p1(2,:)./p1(3,:);
p1(3,:) = p1(3,:)./p1(3,:);
p1(1:2,:) = p1(1:2,:) + 2.0*randn(2,N); % Add a little noise

I1 = zeros(400,400);
figure, imshow(I1, []);
hold on
plot(p1(1,:), p1(2,:), 'w.-');

% Project points onto image2
H_w_c2 = inv(H_c2_w);
Mext = H_w_c2(1:3,:);
p2 = K*Mext*P_w;
p2(1,:) = p2(1,:)./p2(3,:);
p2(2,:) = p2(2,:)./p2(3,:);
p2(3,:) = p2(3,:)./p2(3,:);
p2(1:2,:) = p2(1:2,:) + 2.0*randn(2,N); % Add a little noise

I2 = zeros(400,400);
figure, imshow(I2, []);
hold on
plot(p2(1,:), p2(2,:), 'w.-');
% Reconstruct points

% First normalize the image points
pn1 = inv(K)*p1;
pn2 = inv(K)*p2;

Pr_w = zeros(3,N); % This holds the reconstructed points
for i=1:N
A = zeros(4,3);  
b = zeros(4,1);

% Camera 1
r11 = H_w_c1(1,1);  r12 = H_w_c1(1,2);  r13 = H_w_c1(1,3);
r21 = H_w_c1(2,1);  r22 = H_w_c1(2,2);  r23 = H_w_c1(2,3);
r31 = H_w_c1(3,1);  r32 = H_w_c1(3,2);  r33 = H_w_c1(3,3);
tx = H_w_c1(1,4);   ty = H_w_c1(2,4);   tz = H_w_c1(3,4);   
x = pn1(1,i);       y = pn1(2,i);
A(1,:) =
A(2,:) =
b(1) =
b(2) =

% Camera 2
r11 = H_w_c2(1,1);  r12 = H_w_c2(1,2);  r13 = H_w_c2(1,3);
r21 = H_w_c2(2,1);  r22 = H_w_c2(2,2);  r23 = H_w_c2(2,3);
r31 = H_w_c2(3,1);  r32 = H_w_c2(3,2);  r33 = H_w_c2(3,3);
tx = H_w_c2(1,4);   ty = H_w_c2(2,4);   tz = H_w_c2(3,4);   
x = pn2(1,i);       y = pn2(2,i);
A(3,:) =
A(4,:) =
b(3) =
b(4) =

x = A\b;  % Solve for (X;Y;Z) point position
Pr_w(:,i) = x;
end

Colorado School of Mines  Computer Vision
```
Pr_w = zeros(3,N);  % This holds the reconstructed points
for i=1:N
    % The equations for image point x,y are:
    % x = (r11*X + r12*Y + r13*Z + tx)/(r31*X + r32*Y + r33*Z + tz)
    % y = (r21*X + r22*Y + r23*Z + ty)/(r31*X + r32*Y + r33*Z + tz)
    % or
    % (r11-x*r31)*X + (r12-x*r32)*Y + (r13-x*r33)*Z = -tx + x*tz
    % (r21-y*r31)*X + (r22-y*r32)*Y + (r23-y*r33)*Z = -ty + y*tz
    % Here, (X;Y;Z) are the unknowns. Put into the form Ax=b.
    A = zeros(4,3);
    b = zeros(4,1);

    % Camera 1
    r11 = H_w_c1(1,1);  r12 = H_w_c1(1,2);  r13 = H_w_c1(1,3);
    r21 = H_w_c1(2,1);  r22 = H_w_c1(2,2);  r23 = H_w_c1(2,3);
    r31 = H_w_c1(3,1);  r32 = H_w_c1(3,2);  r33 = H_w_c1(3,3);
    tx = H_w_c1(1,4);   ty = H_w_c1(2,4);   tz = H_w_c1(3,4);
    x = pn1(1,i);       y = pn1(2,i);
    A(1,:) = [ r11-x*r31    r12-x*r32   r13-x*r33 ];
    A(2,:) = [ r21-y*r31    r22-y*r32   r23-y*r33 ];
    b(1) = -tx + x*tz;
    b(2) = -ty + y*tz;

    % Camera 2
    r11 = H_w_c2(1,1);  r12 = H_w_c2(1,2);  r13 = H_w_c2(1,3);
    r21 = H_w_c2(2,1);  r22 = H_w_c2(2,2);  r23 = H_w_c2(2,3);
    r31 = H_w_c2(3,1);  r32 = H_w_c2(3,2);  r33 = H_w_c2(3,3);
    tx = H_w_c2(1,4);   ty = H_w_c2(2,4);   tz = H_w_c2(3,4);
    x = pn2(1,i);       y = pn2(2,i);
    A(3,:) = [ r11-x*r31    r12-x*r32   r13-x*r33 ];
    A(4,:) = [ r21-y*r31    r22-y*r32   r23-y*r33 ];
    b(3) = -tx + x*tz;
    b(4) = -ty + y*tz;

    x = A\b;  % Solve for (X;Y;Z) point position
    Pr_w(:,i) = x;
end
```
Display reconstructed points

% Display reconstructed 3D points
figure(1);
hold on
plot3(Pr_w(1,:), Pr_w(2,:), Pr_w(3,:), 'ro-');

- Ground truth is shown in blue
- Reconstructed is shown in red