Colorado School of Mines

Computer Vision

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

- We have seen how to compute pose, from 2D-3D point correspondences, using non-linear least squares
  - This gives the most accurate results; however, it requires a good initial guess

- Now we will look at how to estimate pose using a linear method, that doesn’t require an initial guess
  - The linear method is called “Direct Linear Transform” (DLT)

- For best results, use the linear method to get an initial guess, then refine it with the nonlinear method
Direct Linear Transform (DLT)

- We can directly solve for the elements of the camera projection matrix.
- Recall the projection of a 3D point $wP$ in the world to a point in the pixel image $(x_{im}, y_{im})$

$$\tilde{p} = KM_{ext}wP$$

$$\tilde{p} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = KM_{ext} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \quad x_{im} = x_1 / x_3, \quad y_{im} = x_2 / x_3$$

- Where the extrinsic parameter matrix is

$$M_{ext} = \begin{pmatrix} c & R & c t_{\text{Worg}} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$$

- And the intrinsic parameter matrix

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- We will solve for the 12 elements of $M_{ext}$ by treating them as independent (of course, they are not independent!)
Normalized Image Coordinates

• We will work with “normalized” image points:
• If we know the intrinsic camera parameter matrix, we can convert the image points to “normalized” image coordinates
  – Origin is in center of image
  – Effective focal length equals 1
  – \( x_{\text{normalized}} = \frac{X}{Z}, \ y_{\text{normalized}} = \frac{Y}{Z} \)

• Then
  – \( \mathbf{p}_{\text{unnormalized}} = \mathbf{K} \mathbf{p}_{\text{normalized}} \)
  – \( \mathbf{p}_{\text{normalized}} = \mathbf{(K)}^{-1} \mathbf{p}_{\text{unnormalized}} \)

• where \( \mathbf{K} \) is the intrinsic parameter matrix

Note – Hartley and Zisserman say that you should precondition in the input values; i.e., translate and scale the image points so that the centroid of the points is at the origin, and the average distance of the points to the origin is equal to \( \sqrt{2} \).
Direct Linear Transform (DLT)

- The projection of a 3D point $W\mathbf{P}$ in the world to a normalized image point is

$$\tilde{\mathbf{p}}_n = \mathbf{M}_{\text{ext}} \mathbf{P} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

or

$$x = \frac{r_{11}X + r_{12}Y + r_{13}Z + t_x}{r_{31}X + r_{32}Y + r_{33}Z + t_z}, \quad y = \frac{r_{21}X + r_{22}Y + r_{23}Z + t_y}{r_{31}X + r_{32}Y + r_{33}Z + t_z}$$

- Multiplying by the denominator

$$r_{11}X + r_{12}Y + r_{13}Z + t_x - x(r_{31}X + r_{32}Y + r_{33}Z + t_z) = 0$$

$$r_{21}X + r_{22}Y + r_{23}Z + t_y - y(r_{31}X + r_{32}Y + r_{33}Z + t_z) = 0$$

- Put into the form $\mathbf{A} \mathbf{x} = 0$

$$\mathbf{A} \mathbf{x} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = 0$$

*How many points do we need to solve for $\mathbf{x}$?*
Direct Linear Transform (DLT)

- The projection of a 3D point $W^p$ in the world to a normalized image point is

$$\tilde{p}_n = M_{ext}P = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

or

$$x = \frac{r_{11}X + r_{12}Y + r_{13}Z + t_x}{r_{31}X + r_{32}Y + r_{33}Z + t_z}, \quad y = \frac{r_{21}X + r_{22}Y + r_{23}Z + t_y}{r_{31}X + r_{32}Y + r_{33}Z + t_z}$$

- Multiplying by the denominator

$$r_{11}X + r_{12}Y + r_{13}Z + t_x - x(r_{31}X + r_{32}Y + r_{33}Z + t_z) = 0$$

$$r_{21}X + r_{22}Y + r_{23}Z + t_y - y(r_{31}X + r_{32}Y + r_{33}Z + t_z) = 0$$

- Put into the form $Ax = 0$

$$Ax = \begin{pmatrix} X & Y & Z & 0 & 0 & 0 & -xX & -xY & -xZ & 1 & 0 & -x \\ 0 & 0 & 0 & X & Y & Z & -yX & -yY & -yZ & 0 & 1 & -y \end{pmatrix} = 0$$

**How many points do we need to solve for $x$?**
DLT Example

% DLT algorithm (direct linear transform)
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create input data

% Create camera matrix
f = 512;      % focal length in pixels
cx = 256;
cy = 256;
K = [ f 0 cx; 0 f cy; 0 0 1 ];   % intrinsic parameter matrix

N = 8;  % Create known 3D points (at least 6)
% P_M = [
%     rand(3,N)-0.5;  % Points within a cube of unit length
%     ones(1,N)
%     ];
P_M = [
    1 -1 1 -1 1 -1 1 -1;  % points on a cube
    1 1 -1 -1 1 1 -1 -1;
    1 1 1 1 -1 -1 -1 -1;
    1 1 1 1 1 1 1 1];
% Create true model-to-camera transform
ax = 0; ay = 20*pi/180; az = -30*pi/180;
tx = 0; ty = 0; tz = 6;
Rx = [ 1 0 0; 0 cos(ax) -sin(ax); 0 sin(ax) cos(ax)];
Ry = [ cos(ay) 0 sin(ay); 0 1 0; -sin(ay) 0 cos(ay)];
Rz = [ cos(az) -sin(az) 0; sin(az) cos(az) 0; 0 0 1];
R_m_c = Rz * Ry * Rx;
H_m_c = [R_m_c [tx;ty;tz]; 0 0 0 1];
disp('Ground truth pose, model to camera:'); disp(H_m_c);

H_c_m = inv(H_m_c);
disp('Ground truth pose, camera to model:'); disp(H_c_m);

% Project points onto image
Mext = H_m_c(1:3, :); % Camera extrinsic matrix
p = K*Mext*P_M;
p(1,:) = p(1,:)./p(3,:);
p(2,:) = p(2,:)./p(3,:);
p(3,:) = p(3,:)./p(3,:);
% Display input data
disp('Known model points:'); disp(P_M);
disp('Measured image points:'); disp(p);
I = zeros(512,512);
imshow(I);
hold on
plot(p(1,:), p(2,:), 'g*');

% Add some noise to the image points
sigma = 5.0;
p(1:2,:) = p(1:2,:) + sigma*randn(2,N);
plot(p(1,:), p(2,:), 'w*');
% Solve for the pose of the model with respect to the camera.

\[ \text{pn} = \text{inv}(K) \ast p; \] % Normalize image points

% Ok, now we have \( \text{pn} = \text{Mext} \ast \text{P}_M \).
% If we know \( \text{P}_M \) and \( \text{pn} \), we can solve for the elements of \( \text{Mext} \).
% The equations for \( x, y \) are:
% \[
% x = \frac{(r11 \ast X + r12 \ast Y + r13 \ast Z + tx)}{(r31 \ast X + r32 \ast Y + r33 \ast Z + tz)}
% \]
% \[
% y = \frac{(r21 \ast X + r22 \ast Y + r23 \ast Z + ty)}{(r31 \ast X + r32 \ast Y + r33 \ast Z + tz)}
% \]
% or
% \[
% r11 \ast X + r12 \ast Y + r13 \ast Z + tx - x \ast r31 \ast X - x \ast r32 \ast Y - x \ast r33 \ast Z - x \ast tz = 0
% \]
% \[
% r21 \ast X + r22 \ast Y + r23 \ast Z + ty - y \ast r31 \ast X - y \ast r32 \ast Y - y \ast r33 \ast Z - y \ast tz = 0
% \]

% Put elements of \( \text{Mext} \) into vector \( w \):
% \[
% w = [r11 \ r12 \ r13 \ r21 \ r22 \ r23 \ r31 \ r32 \ r33 \ tx \ ty \ tz]
% \]
% We then have \( A \ast x = 0 \). The rows of \( A \) are:
% \[
% \begin{bmatrix}
% X & Y & Z & 0 & 0 & 0 & -x \ast X & -x \ast Y & -x \ast Z & 1 & 0 & -x
% \end{bmatrix}
% \]
% \[
% \begin{bmatrix}
% 0 & 0 & 0 & X & Y & Z & -y \ast X & -y \ast Y & -y \ast Z & 1 & 0 & -y
% \end{bmatrix}
% \]

\[ A = \text{zeros}(N,12); \]
\[ \text{for } i=1:N \]
\[ X = \text{P}_M(1,i); \quad Y = \text{P}_M(2,i); \quad Z = \text{P}_M(3,i); \]
\[ x = \text{pn}(1,i); \quad y = \text{pn}(2,i); \]
\[ A( 2*(i-1)+1, :) = [ X \ Y \ Z \ 0 \ 0 \ 0 \ -x \ast X \ -x \ast Y \ -x \ast Z \ 1 \ 0 \ -x ]; \]
\[ A( 2*(i-1)+2, :) = [ 0 \ 0 \ 0 \ X \ Y \ Z \ -y \ast X \ -y \ast Y \ -y \ast Z \ 0 \ 1 \ -y ]; \]
\[ \text{end} \]
Solving a System of Homogeneous Equations

• We want to solve a system of $m$ linear equations in $n$ unknowns, of the form $Ax = 0$
  – Note that any scaled version of $x$ is also a solution ($x=0$ is not interesting)

• The solution $x$ is the eigenvector corresponding to the only zero eigenvalue of $A^TA$

• Equivalently, we can take the SVD of $A$; i.e., $A = U D V^T$
  – And $x$ is the column of $V$ corresponding to the zero singular value of $A$
  – (Since the columns are ordered, this is the rightmost column of $V$)
D LT Example

% 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  t]
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 now have the camera extrinsic matrix M (up to a scale factor).
• Now, we need to extract the rotation and translation from M.
Extracting translation

• The projection matrix is a 3x4 matrix

\[ M = \begin{bmatrix} c^c R & c^c t_{morg} \end{bmatrix} \]

rotation matrix, model to camera

origin of model with respect to camera

• Recall (from lecture on 3D-3D transformations)

\[ c^c t_{morg} = -m^c R \cdot m^m t_{corg} \]

i.e., the origin of the model frame with respect to the camera frame is the (rotated) negative of the origin of the camera frame with respect to the model frame

• So

\[ M = c^c R \begin{bmatrix} I_{3x3} & -m^m t_{corg} \end{bmatrix} \]
Extracting translation (continued)

• Now if we multiply $\mathbf{M}$ by the vector representing the camera origin with respect to the model, we get zero:

$$
\begin{bmatrix}
m \mathbf{t}_{corg} \\
1
\end{bmatrix} =
\begin{bmatrix}
c \\
\\
m -m \\
1
\end{bmatrix}
\begin{bmatrix}
\mathbf{R} \\
\mathbf{I}_{3 \times 3}
\end{bmatrix}
\begin{bmatrix}
\mathbf{m} \\
1
\end{bmatrix}
= 0
$$

• So solve the system $\mathbf{M}\mathbf{X} = 0$; then scale the result so that the 4th element = 1

% We can find the camera center, $\mathbf{t}_{corg\_m}$ by solving the equation $\mathbf{M}\mathbf{X} = 0$.  
% To see this, write $\mathbf{M} = [\mathbf{R}_{m\_c} \mathbf{t}_{morg\_c}]$.  But $\mathbf{t}_{morg\_c} = -\mathbf{R}_{m\_c} * \mathbf{t}_{corg\_m}$.  
% So $\mathbf{M} = \mathbf{R}_{m\_c}\times [I \quad -\mathbf{t}_{corg\_m}]$.  And if we multiply $\mathbf{M}$ times $\mathbf{t}_{corg\_m}$, we get  
% $\mathbf{R}_{m\_c}\times [I \quad -\mathbf{t}_{corg\_m}] \times [\mathbf{t}_{corg\_m} ; 1] = 0$.  
[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
Extracting the rotation

• The leftmost 3x3 portion of $\mathbf{M}$ represents the rotation

$$\mathbf{M} = \begin{bmatrix}
  c & R & c t_{morig}
\end{bmatrix}$$

• However, that 3x3 submatrix of $\mathbf{M}$ (as estimated) may not be a valid rotation matrix:
  – A valid rotation matrix is orthonormal (ie its rows and columns are unit vectors and are orthogonal to each other)
  – A valid rotation matrix has determinant = +1 (i.e., it is a right-handed coordinate system)

• To get a valid rotation matrix, we will do “QR” decomposition
QR Decomposition

• Any real square matrix $A$ may be decomposed as $A = QR$, where
  – $Q$ is an orthonormal matrix
  – $R$ is an upper triangular matrix

\[
\begin{pmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{pmatrix} =
\begin{pmatrix}
q_{11} & q_{12} & q_{13} \\
q_{21} & q_{22} & q_{23} \\
q_{31} & q_{32} & q_{33}
\end{pmatrix}
\begin{pmatrix}
          & r_{12} & r_{13} \\
          & r_{22} & r_{23} \\
0 & r_{32} & r_{33}
\end{pmatrix}
\]

\[
A = Q R
\]

• Note the unfortunate clash of terminology ... we have been using “$R$” to represent a rotation matrix. To avoid this, let’s use $B$ to represent the triangular matrix

\[
\begin{pmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{pmatrix} =
\begin{pmatrix}
q_{11} & q_{12} & q_{13} \\
q_{21} & q_{22} & q_{23} \\
q_{31} & q_{32} & q_{33}
\end{pmatrix}
\begin{pmatrix}
          & b_{12} & b_{13} \\
          & b_{22} & b_{23} \\
0 & b_{32} & b_{33}
\end{pmatrix}
\]

\[
A = QB
\]
Extracting the rotation

• Assume that the leftmost 3x3 portion of M is the rotation, but multiplied by some scaling matrix (this could be the intrinsic camera parameter matrix)

\[ M_{1:3,1:3} = KR \]

• The transpose is

\[ (M_{1:3,1:3})^T = (KR)^T = R^T K^T \]

• We take the “QR” decomposition to get

\[ R^T K^T = QB \]

• So “Q” is the transpose of the rotation matrix that we want
% 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);
Display Predicted Points

- Using the estimated pose, and the known 3D points, predict where the points would project onto the image (and display that)

```matlab
% 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*');
```
Pose Error

- We want to quantify the error between the estimated pose and the (known) ground truth pose.
- We can compute the transformation from the true model frame \( (m_{true}) \) to the estimated model pose \( (m_{est}) \):
  \[
  m_{true} H = \left( m_{true} H \right) \left( camera H \right)
  \]
- Then to quantify the error, use:
  - Translation error – just take the length of the translation vector.
  - Rotation error - find the (axis,angle) equivalent of the rotation matrix, and then use the angle.
Equivalent Angle-Axis

• A general rotation can be expressed as a rotation $\theta$ about an axis $k$

$$R_k(\theta) = \begin{pmatrix}
k_xk_xv\theta + c\theta & k_xk_yv\theta - k_zs\theta & k_xk_zv\theta + k ys\theta \\
k_xk_yv\theta + k_zs\theta & k_yk_yv\theta + c\theta & k_yk_zv\theta - k_xs\theta \\
k_xk_zv\theta - k_y s\theta & k_yk_zv\theta + k_xs\theta & k_zk_zv\theta + c\theta
\end{pmatrix}$$

where

$$c\theta = \cos\theta, \quad s\theta = \sin\theta, \quad v\theta = 1 - \cos\theta$$

$$\mathbf{k} = \begin{pmatrix} k_x \\ k_y \\ k_z \end{pmatrix}$$

• The inverse solution (i.e., given a rotation matrix, find $k$ and $\theta$):

$$\theta = \acos \left( \frac{r_{11} + r_{22} + r_{33} - 1}{2} \right)$$

$$\mathbf{k} = \frac{1}{2 \sin\theta} \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

Note that (-$k$, -$\theta$) is also a solution.
DLT Example – Quantify Pose Error

%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate error (from ground truth)

Hdiff = H_c_m * Hestimated_m_c; % Transformation error
Rdiff = Hdiff(1:3,1:3); % Rotation between our answer and ground truth
ang = acos( (trace(Rdiff)-1)/2 );
fprintf('Rotation error (degrees): %f\n', ang*180/pi);
tdiff = Hdiff(1:3, 4);
fprintf('Translation error: %f\n', norm(tdiff));
DLT for Motion Capture

• We can use the DLT method for tracking markers for motion capture applications (sports, animation)

• Approach:
  – Set up a calibration grid with known target points
  – Determine the camera projection matrices for multiple cameras

• Run time
  – Each marker must be seen by more than one camera
  – Each marker’s 3D position can be reconstructed from the corresponding image points

A dancer wearing a suit used in an optical motion capture system (from Wikipedia article on motion capture)
DLT for Reconstruction

• 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

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

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

• Again, rearrange to put into the form \(A \, \mathbf{x} = 0\) 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 \((r_{11}, r_{12}, \ldots, t_y, t_z)\)