Image Processing

Cosimo Distante

Lecture: Alignment (cont’d)
Other Pose estimation techniques

- Non linear solution
- Linear solution with DLT
Model-based Pose Estimation

• Problem Statement
  – Given
    • We have an image of an object
    • We know the model geometry of the object (specifically, the location of features on the object)
    • We have found the corresponding features in the image
  – Find
    • The position and orientation (pose) of the object with respect to the camera

• Assumptions
  – Object is rigid (so 6 only degrees of freedom)
  – Camera intrinsic parameters are known

• We will find the pose that minimizes the squared error of the predicted locations of the image features, to the measured locations

\[ \text{# points needed?} \]

\[ \text{we want } C_M H \]
Least Squares Pose Estimation

- Let $y = f(x)$
  - $x$ is a vector of the unknown pose parameters
  - $f$ is a function that returns the predicted image points $y$, given the pose $x$
  - $y_0$ is a vector of the actual observed image points

- We want to find $x$ to minimize $E = |f(x) - y_0|^2$

- Algorithm:
  1. We start with a guess for $x$, call it $x_0$
  2. Compute $y = f(x)$. Residual error is $dy = y - y_0$
  3. Calculate Jacobian of $f$, $J = \left[ \frac{\partial f}{\partial x} \right]$, and evaluate it at $x$. We now have $dy = J \cdot dx$
  4. Solve for $dx$ using pseudo inverse $dx = (J^T J)^{-1} J^T dy$
  5. Set $x = x + dx$
  6. Repeat steps 2-5 until convergence (no more change in $x$)
Recall Perspective Projection

- Projection of a 3D point \( wP \) in the world to a point in the pixel image \((x_{im}, y_{im})\)

\[
\tilde{p} = K M_{ext}^{w} P = K M_{ext}^{w} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \quad x_{im} = x_1 / x_3, \quad y_{im} = x_2 / x_3
\]

- Where the extrinsic parameter matrix is

\[
M_{ext} = \begin{pmatrix} c_C & R & c_{t_{Worg}} \\
W & R & t_{Worg} \\
X & Y & Z \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_X \\
r_{21} & r_{22} & r_{23} & t_Y \\
r_{31} & r_{32} & r_{33} & t_Z \end{pmatrix}
\]

- And the intrinsic parameter matrix

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

- Or, if we use “model” instead of “world” frame for the point:

\[
\tilde{p} = K M_{ext}^{M} P = K \begin{pmatrix} c_C & R & c_{t_{Morg}} \end{pmatrix}^{M} P
\]
Recall XYZ fixed angle convention

\[ R = R_z \cdot R_y \cdot R_x, \text{ where} \]

\[
R_z = \begin{pmatrix}
\cos \theta_z & -\sin \theta_z & 0 \\
\sin \theta_z & \cos \theta_z & 0 \\
0 & 0 & 1
\end{pmatrix}
\quad R_y = \begin{pmatrix}
\cos \theta_y & 0 & \sin \theta_y \\
0 & 1 & 0 \\
-\sin \theta_y & 0 & \cos \theta_y
\end{pmatrix}
\quad R_x = \begin{pmatrix}
1 & 0 & 0 \\
0 & \cos \theta_x & -\sin \theta_x \\
0 & \sin \theta_x & \cos \theta_x
\end{pmatrix}
\]

• Matlab

\[
R_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\alpha) & -\sin(\alpha) \\
0 & \sin(\alpha) & \cos(\alpha)
\end{bmatrix};
R_y = \begin{bmatrix}
\cos(\beta) & 0 & \sin(\beta) \\
0 & 1 & 0 \\
-\sin(\beta) & 0 & \cos(\beta)
\end{bmatrix};
R_z = \begin{bmatrix}
\cos(\gamma) & -\sin(\gamma) & 0 \\
\sin(\gamma) & \cos(\gamma) & 0 \\
0 & 0 & 1
\end{bmatrix};
R = R_z \cdot R_y \cdot R_x
\]

*Note – in general, the “angle-axis” convention would be better to use than XYZ angles*
Function to project one point

• Write a function to project a 3D point \( P_M \) in model coordinates to image point \( p \), given the model-to-camera pose \( x = (ax,ay,az,tx,ty,tz) \)
  – \( P_M = [X;Y;Z;1] \) is the input point
  – \( x = [ax;ay;az;tx;ty;tz] \) is the vector of model-to-camera pose parameters
  – \( K \) = intrinsic camera matrix
  – \( p = [x;y] \) is the output point

```matlab
function p = fProject(x, P_M, K)
% Project 3D point onto image

% Get pose params
ax = x(1); ay = x(2); az = x(3);
tx = x(4); ty = x(5); tz = x(6);

% Rotation matrix, model to camera
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 = Rz * Ry * Rx;

% Extrinsic camera matrix
Mext = [ R [tx;ty;tz] ];

% Project point
ph = K*Mext*P_M;
ph = ph/ph(3);

p = ph(1:2);
return
```
Function to transform a set of points

- Now modify the function to transform a set of points
  - $P_M$ = is a set of input points
  - $x = [ax; ay; az; tx; ty; tz]$ is the pose
  - $K =$ intrinsic camera matrix
  - $p = [x_1; y_1; x_2; y_2; \ldots]$ are the output points

$$MP = \begin{pmatrix} X_1 & X_2 & \cdots & X_N \\ Y_1 & Y_2 & \cdots & Y_N \\ Z_1 & Z_2 & \cdots & Z_N \\ 1 & 1 & \cdots & 1 \end{pmatrix} \quad p = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_N \\ y_N \end{pmatrix}$$

function $p = fProject(x, P_M, K)$
Function to transform a set of points

```matlab
function p = fProject(x, P_M, K)
% Project 3D points onto image

% Get pose params
ax = x(1); ay = x(2); az = x(3);
tx = x(4); ty = x(5); tz = x(6);

% Rotation matrix, model to camera
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 = Rz * Ry * Rx;

% Extrinsic camera matrix
Mext = [ R [tx;ty;tz] ];

% Project points
ph = K*Mext*P_M;

% Divide through 3rd element of each column
ph(1,:) = ph(1,:)./ph(3,:);
ph(2,:) = ph(2,:)./ph(3,:);
ph = ph(1:2,:); % Get rid of 3rd row

p = reshape(ph, [], 1); % reshape into 2Nx1 vector
return
```
Example

Focal length in pixels: 715

Image center (x,y): (354, 245)
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 ];

% 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

y0 = [ 183; 147; % 1
       350; 133; % 2
       454; 144; % 3
       176; 258; % 4
       339; 275; % 5
       444; 286 ]; % 6

% Make an initial guess of the pose [ax ay az tx ty tz]
x = [1.5; -1.0; 0.0; 0; 0; 30];

% Get predicted image points by substituting in the current pose
y = fProject(x, P_M, K);

for i=1:2:length(y)
    rectangle('Position', [y(i)-8 y(i+1)-8 16 16], 'FaceColor', 'r');
end
Computing Jacobian Numerically

- We approximate the derivatives

\[
\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \epsilon \hat{u}_i) - f(x)}{\epsilon}
\]

\[
J = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_M} \\
\frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_M} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_M}
\end{bmatrix}
\]

- Matlab code:

```matlab
y = fProject(x,P_M,K);

e = 0.000001;  % a tiny number
J(:,1) = ( fProject(x+[e;0;0;0;0;0],P_M,K) - y )/e;
J(:,2) = ( fProject(x+[0;e;0;0;0;0],P_M,K) - y )/e;
J(:,3) = ( fProject(x+[0;0;e;0;0;0],P_M,K) - y )/e;
J(:,4) = ( fProject(x+[0;0;0;e;0;0],P_M,K) - y )/e;
J(:,5) = ( fProject(x+[0;0;0;0;e;0],P_M,K) - y )/e;
J(:,6) = ( fProject(x+[0;0;0;0;0;e],P_M,K) - y )/e;
```
We have

- \( y_0 \) = observations or measurements
- \( x_0 \) = a guess for \( x \)
- \( y = f(x) \) is a non linear function

1. Initialize \( x \) to \( x_0 \)
2. Compute \( y = f(x) \).
   Residual error is \( dy = y - y_0 \)
3. Calculate Jacobian of \( f \), evaluate it at \( x \). We now have \( dy = J \, dx \)
4. Solve for \( dx \) using pseudo inverse \( dx = (J^TJ)^{-1}J^T \, dy \)
5. Set \( x <= x + dx \)
6. Repeat steps 2-5 until convergence (no more change in \( x \))
for i=1:10
    fprintf('nIteration %d\nCurrent pose:\n', i);
    disp(x);

    % Get predicted image points
    y = fProject(x, P_M, K);
    imshow(I, [])
    for i=1:2:length(y)
        rectangle('Position', [y(i)-8 y(i+1)-8 16 16], ...
                   'FaceColor', 'r');
    end
    pause(1);
    % Estimate Jacobian
    e = 0.00001; % a tiny number
    J(:,1) = ( fProject(x+[e;0;0;0;0;0],P_M,K) - y )/e;
    J(:,2) = ( fProject(x+[0;e;0;0;0;0],P_M,K) - y )/e;
    J(:,3) = ( fProject(x+[0;0;e;0;0;0],P_M,K) - y )/e;
    J(:,4) = ( fProject(x+[0;0;0;e;0;0],P_M,K) - y )/e;
    J(:,5) = ( fProject(x+[0;0;0;0;e;0],P_M,K) - y )/e;
    J(:,6) = ( fProject(x+[0;0;0;0;0;e],P_M,K) - y )/e;

    % Error is observed image points - predicted image points
    dy = y0 - y;
    fprintf('Residual error: %f\n', norm(dy));

    % Ok, now we have a system of linear equations dy = J dx
    % Solve for dx using the pseudo inverse
    dx = pinv(J) * dy;

    % Stop if parameters are no longer changing
    if abs( norm(dx)/norm(x) ) < 1e-6
        break;
    end

    x = x + dx; % Update pose estimate
end
Overlaying Graphical Model

- As a check, overlay a graphical model onto the image

- If you don’t have a model, you can display the model’s coordinate axes

```matlab
% Draw coordinate axes onto the image. Scale the length of the axes
% according to the size of the model, so that the axes are visible.
W = max(P_M,[],2) - min(P_M,[],2); % Size of model in X,Y,Z
W = norm(W); % Length of the diagonal of the bounding box

u0 = fProject(x, [0;0;0;1], K); % origin
uX = fProject(x, [W/5;0;0;1], K); % unit X vector
uY = fProject(x, [0;W/5;0;1], K); % unit Y vector
uZ = fProject(x, [0;0;W/5;1], K); % unit Z vector

line([u0(1) uX(1)], [u0(2) uX(2)], 'Color', 'r', 'LineWidth', 3);
line([u0(1) uY(1)], [u0(2) uY(2)], 'Color', 'g', 'LineWidth', 3);
line([u0(1) uZ(1)], [u0(2) uZ(2)], 'Color', 'b', 'LineWidth', 3);

% Also print the pose onto the image.
ax=1.55 ay=-0.83 az=0.10 tx=0.9 ty=3.0 tz=18.3
text(30,450,sprintf('ax=%.2f ay=%.2f az=%.2f tx=%.1f ty=%.1f tz=%.1f', ...
    x(1), x(2), x(3), x(4), x(5), x(6)), ...
    'BackgroundColor', 'w', 'FontSize', 15);"
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

-> Can determine the pose of the model

single (calibrated) camera
3D->2D 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 \((x_{im}, y_{im})\)

\[
\tilde{p} = KM_{\text{ext}}^w P \quad \tilde{p} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = KM_{\text{ext}}^w \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \quad x_{im} = x_1 / x_3, \quad y_{im} = x_2 / x_3,
\]

- Where the extrinsic parameter matrix is

\[
M_{\text{ext}} = \begin{pmatrix} c_w R & c_t_{\text{Worg}} \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}
\]

- 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 \( M_{\text{ext}} \) 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
  - \( x_{\text{normalized}} = \frac{x}{Z}, \quad y_{\text{normalized}} = \frac{y}{Z} \)

- Then
  - \( p_{\text{unnormalized}} = K \ p_{\text{normalized}} \)
  - \( p_{\text{normalized}} = (K)^{-1} \ p_{\text{unnormalized}} \)

- where \( K \) is the intrinsic parameter matrix

Note – Hartley and Zisserman say that you should precondition in the input values; ie., 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 $^wP$ 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 $A x = 0$

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

How many points do we need to solve for $x$?
Direct Linear Transform (DLT)

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

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

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 $\mathbf{A} \mathbf{x} = 0$

$$\mathbf{A} \mathbf{x} = \begin{pmatrix} X & Y & Z & 0 & 0 & 0 & -xX & -xY & -xZ & 1 & 0 & -x \\ 0 & 0 & 0 & X & Y & Z & -yX & -yY & -yZ & 0 & 1 & -y \end{pmatrix} \begin{pmatrix} r_{11} \\ r_{12} \\ r_{13} \\ r_{21} \\ r_{22} \\ r_{23} \\ r_{31} \\ r_{32} \\ r_{33} \\ t_x \\ t_y \\ t_z \end{pmatrix} = \mathbf{0}$$

**How many points do we need to solve for $\mathbf{x}$?**
% 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;
    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,:);
DLT Example

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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.

\[ \text{pn} = \text{inv}(K) \ast \text{p}; \]  % 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 \( Ax = 0 \). The rows of \( A \) are:
% \[ X \ Y \ Z \ 0 \ 0 \ 0 \ -x \ast X \ -x \ast Y \ -x \ast Z \ 1 \ 0 \ -x \]
% \[ 0 \ 0 \ 0 \ X \ Y \ Z \ -y \ast X \ -y \ast Y \ -y \ast Z \ 1 \ 0 \ -y \]

\[ A = \text{zeros}(N,12); \]
\[ \text{for } i=1:N \]
\[ X = \text{P_M}(1,i); \quad Y = \text{P_M}(2,i); \quad Z = \text{P_M}(3,i); \]
\[ x = \text{pn}(1,i); \quad y = \text{pn}(2,i); \]
\[ A( 2 \ast (i-1)+1, :) = [ X \ Y \ Z \ 0 \ 0 \ 0 \ -x \ast X \ -x \ast Y \ -x \ast Z \ 1 \ 0 \ -x ]; \]
\[ A( 2 \ast (i-1)+2, :) = [ 0 \ 0 \ 0 \ X \ Y \ Z \ -y \ast X \ -y \ast Y \ -y \ast Z \ 0 \ 1 \ -y ]; \]
\[ \text{end} \]
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 $A$; ie., $A = U \, D \, V^T$
  – 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 $V$)
% 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

\[
M = \begin{bmatrix}
  cR & c t_{\text{morg}} \\
  m & m t_{\text{corg}}
\end{bmatrix}
\]

rotation matrix, model to camera
origin of model with respect to camera

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

\[
M = \begin{bmatrix}
  cR & -m t_{\text{corg}} \\
  m & m t_{\text{corg}}
\end{bmatrix}
\]
Extracting translation cont’d

• Now if we multiply \( M \) by the vector representing the camera origin with respect to the model, we get zero:

\[
M \begin{pmatrix} m \mathbf{t}_{\text{corg}} \\ 1 \end{pmatrix} = c \mathbf{R} \begin{bmatrix} I_{3x3} & -m \mathbf{t}_{\text{corg}} \end{bmatrix} \begin{pmatrix} m \mathbf{t}_{\text{corg}} \\ 1 \end{pmatrix} = 0
\]

• So solve the system \( MX=0 \); then scale the result so that the 4\(^{th}\) element = 1

% We can find the camera center, \( \mathbf{t}_{\text{corg}} \) by solving the equation \( MX=0 \).% To see this, write \( M = [\mathbf{R}_{m_c} \mathbf{t}_{\text{morg}}] \). But \( \mathbf{t}_{\text{morg}} = -\mathbf{R}_{m_c} \times \mathbf{t}_{\text{corg}} \).% So \( M = \mathbf{R}_{m_c} \times [I \ -\mathbf{t}_{\text{corg}}] \). And if we multiply \( M \) times \( \mathbf{t}_{\text{corg}} \), we% get \( \mathbf{R}_{m_c} \times [I \ -\mathbf{t}_{\text{corg}}] \times [\mathbf{t}_{\text{corg}}; 1] = 0 \).
% 
% \[
[U, D, V] = \text{svd}(M);
% \mathbf{t}_{\text{corg}} = V(:,\text{end}); \quad \% \text{Get last column of } V
% \mathbf{t}_{\text{corg}} = \mathbf{t}_{\text{corg}} / \mathbf{t}_{\text{corg}}(4); \quad \% \text{Divide through by last element}
Extracting the rotation

- The leftmost 3x3 portion of $M$ represents the rotation
  \[ M = \begin{bmatrix} c^R & c^R t_{morg} \end{bmatrix} \]

- However, that 3x3 submatrix of $M$ (as estimated) may not be a valid rotation matrix:
  - A valid rotation matrix is orthonormal (i.e., 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 $\mathbf{A}$ may be decomposed as $\mathbf{A} = \mathbf{QR}$, where
  - $\mathbf{Q}$ is an orthonormal matrix
  - $\mathbf{R}$ is an upper triangular matrix

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

- $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{R}$

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

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

- $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{B}$

$\mathbf{A} = \mathbf{QB}$

Hoff
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

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

• We take the “QR” decomposition to get

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

• So “Q” is the transpose of the rotation matrix that we want
Extracting the rotation

% 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 ($m_{est}$)

$$
\begin{align*}
\begin{bmatrix}
  m_{true} \\
  m_{est}
\end{bmatrix}
H =
\begin{bmatrix}
  m_{true} \\
  \text{camera}
\end{bmatrix}
H
\begin{bmatrix}
  \text{camera} \\
  m_{est}
\end{bmatrix}
H
\end{align*}
$$

• 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 $k$

$$R_k(\theta) = \begin{pmatrix}
  k_x k_x \nu \theta + c \theta & k_x k_y \nu \theta - k_z s \theta & k_x k_z \nu \theta + k_y s \theta \\
  k_x k_y \nu \theta + k_z s \theta & k_y k_y \nu \theta + c \theta & k_y k_z \nu \theta - k_x s \theta \\
  k_x k_z \nu \theta - k_y s \theta & k_y k_z \nu \theta + k_x s \theta & k_z k_z \nu \theta + c \theta
\end{pmatrix}$$

where

$$c \theta = \cos \theta, \ s \theta = \sin \theta, \ \nu \theta = 1 - \cos \theta$$

$$k = (k_x, k_y, k_z)^T$$

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

• The product of the unit vector $k$ and angle $\theta$, $\omega = \hat{\theta} k = (\omega_x, \omega_y, \omega_z)$ is a minimal representation for a 3D rotation

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

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

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

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

• 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 \((r_{11}, r_{12}, ..., t_y, t_z)\)
3D->2D

Algorithm 1 (The DLT calibration) Given at least six 3D-2D point correspondences \( \mathbf{x}_i \leftrightarrow \mathbf{u}_i \) for \( i = 1, \ldots, n \) and \( n \geq 6 \), compute the intrinsic parameters \( \mathbf{K} \) of the camera and the rotation \( \mathbf{R} \) and the translation \( \mathbf{t} \) of the camera with respect to the points \( \mathbf{x}_i \).

1. Compute a 2D similarity transformation \( \mathbf{T}_u \) such that the points \( \mathbf{u}_i \) are translated to its centroid and re-scaled so that the average distance equals to \( \sqrt{2} \). Do the same to compute a 3D similarity transformation \( \mathbf{T}_x \) for \( \mathbf{x}_i \) so that the average distance equals to \( \sqrt{3} \).
2. Apply \( \tilde{\mathbf{u}}_i = \mathbf{T}_u \mathbf{u}_i \) and \( \tilde{\mathbf{x}}_i = \mathbf{T}_x \mathbf{x}_i \).
3. Form the \( \tilde{\mathbf{A}}_{2n \times 12} \) matrix with normalized points \( \tilde{\mathbf{u}}_i \) and \( \tilde{\mathbf{x}}_i \).
4. Solve for \( \tilde{\mathbf{p}}_{12} \) by taking the singular vector corresponding to the smallest singular value of \( \tilde{\mathbf{A}}_{2n \times 12} \).
\[
\begin{pmatrix}
    x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -u_i x_i & -u_i y_i & -u_i z_i & -u_i \\
    0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -v_i x_i & -v_i y_i & -v_i z_i & -v_i
\end{pmatrix}
\begin{pmatrix}
    \mathbf{p}_{12}
\end{pmatrix} = \mathbf{0},
\]
\[
\tilde{\mathbf{A}}_{2n \times 12} \tilde{\mathbf{p}}_{12} = \mathbf{0}.
\]
5. Convert \( \tilde{\mathbf{p}}_{12} \) into the matrix \( \tilde{\mathbf{P}} \).
6. Undo the normalization by \( \mathbf{P} = \mathbf{T}_x^{-1} \tilde{\mathbf{P}} \mathbf{T}_u \).
7. Decompose \( \mathbf{P} \) to obtain the intrinsic parameters in \( \mathbf{K} \) and the extrinsic parameters \( \mathbf{R} \) and \( \mathbf{t} \).

The solution is unique.