Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science
http://inside.mines.edu/~whoff/
Pose Estimation
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
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 $f$, $J = \frac{\partial f}{\partial x}$, and evaluate it at $x$. We now have $dy = Jdx$
  4. Solve for $dx$ using pseudo inverse $dx = (J^TJ)^{-1}J^Tdy$
  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 $(x_{im}, y_{im})$

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

- Where the extrinsic parameter matrix is

\[
M_{ext} = \begin{pmatrix}
C^w R & C^w t_{Worg}
\end{pmatrix} = \begin{pmatrix}
r_{11} & r_{12} & r_{13} & t_X \\
r_{21} & r_{22} & r_{23} & t_Y \\
r_{31} & r_{32} & r_{33} & t_Z
\end{pmatrix}
\]

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

\[ \tilde{p} = K M_{ext}^M P = K \begin{pmatrix}
C^M R & C^M t_{Morg}
\end{pmatrix}^M P \]

- And the intrinsic parameter matrix

\[
K = \begin{pmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{pmatrix}
\]
Recall XYZ fixed angle convention

- \( R = R_z \cdot R_y \cdot R_x \), 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

\[
R_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(ax) & -\sin(ax) \\
0 & \sin(ax) & \cos(ax)
\end{bmatrix};
R_y = \begin{bmatrix}
\cos(ay) & 0 & \sin(ay) \\
0 & 1 & 0 \\
-\sin(ay) & 0 & \cos(ay)
\end{bmatrix};
R_z = \begin{bmatrix}
\cos(az) & -\sin(az) & 0 \\
\sin(az) & \cos(az) & 0 \\
0 & 0 & 1
\end{bmatrix};
R = R_z \cdot R_y \cdot R_x
\]

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
  - \( p = [x1; y1; x2; y2; \ldots] \) are the output points

\[
function \ p = fProject(x, P_M, K)
\]
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

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

- 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;
```
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);

    % Get data
    y0 = observations or measurements
    x0 = a guess for x
    y = f(x) is a non linear function

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('
Iteration %d
Current pose:
', 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
', 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

% 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);