CSC487/2503 – Computational Vision Tutorial: Oct 15 '01

- Linear parameterization for lines: $\vec{a}$-space formulation
- Assignment 2 walk through
  - Casting votes in the Hough space is similar to line drawing: http://www.dgp.toronto.edu/~ ah/csc418/fall_2001/notes/lines.html
- Extracting Point Features: Corner detector
  - Trucco and Verri, pg. 82-85
  - *Parametric Feature Detection*, S. Baker, S. K. Nayar, and H. Murase, **International Journal of Computer Vision, March 1998**. http://www.cs.columbia.edu/CAVE/research/publications/appearance_matching.html
- Questions?

Line parameterization

(θ, c) formulation

Consider the standard image coordinates with x to the right, y downwards and the origin at the top left-hand corner. Imagine a line L in the image and a vector perpendicular to the line, also called the normal vector $\vec{n}$. For any point $\vec{x} = (x, y)^T$ on the line L, the inner product given by $\vec{n} \cdot \vec{x}$ measures the projection of $\vec{x}$ in the direction of $\vec{n}$. Taking inner product is similar to dropping a perpendicular from the origin on to the line L in the direction of the normal vector $\vec{n}$ and measuring its length. All the points on the line have the *same* perpendicular distance from the origin. Let c denote the length of this perpendicular. Then, the constraint defining an image line L is

$$\vec{n} \cdot \vec{x} + c = 0. \tag{1}$$

If we consider $\vec{n}$ to be a unit vector, say $\vec{n}(\theta) = (\cos(\theta), \sin(\theta))^T$, then there are two unknows to derive for each image line L: θ and c. Here, θ is the angle made by the normal vector $\vec{n}$ with the x-axis. To get some intuition for θ consider an image line L that is vertical. It will have a normal vector $\vec{n}$ that is parallel to the x-axis and hence $\theta = 0$. Similarly, for a horizontal image line the corresponding normal vector has $\theta = 90$. Also, if we are measuring the properties of a line drawn in a *sub-window*, whose origin (the top left-hand corner) is given by $\vec{x}_0$, then the line constraint is given by

$$\vec{n} \cdot (\vec{x} - \vec{x}_0) + c = 0. \tag{2}$$

Some line finding approaches choose x_0 to be in the middle of the selected region so as to reduce round-off errors. As we saw in the class, this formulation specifies lines in the image space as curves in the Hough space. Next we look at a formulation which establishes a nice duality: lines in image space are points in the Hough space and vice versa.

(a_1, a_2) Dual-space/Linear formulation

What if the normal vector $\vec{n}$ is rewritten as a linear combination of two **pre-specified orthogonal** vectors: $\vec{n}_0$ and $\vec{t}_0$. In particular, write

$$\vec{n}(a) = \vec{n}_0 + a\vec{t}_0 \tag{3}$$

Notice there is only one unknown a.

Intuition: imagine $\vec{n}_0$ and $\vec{t}_0$ as two orthogonal unit vectors drawn inside a unit radius circle around the origin. Consider a line M that is a tangent to the circle and perpendicular to $\vec{n}_0$. The points on line M each correspond to the orientations of line normals $\vec{n}$ and they can all be described by the expression given above. In particular, what if $a = \pm 1$? It corresponds to $\vec{n}(a)$ that is $\pi/4$ away from $\vec{n}_0$. Notice what happens in the limit as $a \rightarrow \pm\infty$. It indicates that $\vec{n}$ is perpendicular to $\vec{n}_0$.

How is $\vec{n}_0$ set? In **lineTutorial.m** a "mean" line orientation θ is used for the definition: $\vec{n}_0(\theta) = (\cos(\theta), \sin(\theta))^T$. Look for the variable:**thetaDeg** which is then converted to **thetaRad** in **plotCstLines.m**. We will experiment with this variable in the next section. The orthogonal vector $\vec{t}_0$ is chosen to satisfy: $\vec{n}_0(\theta) \cdot \vec{t}_0(\theta) = 0$.

The line constraint can now be written in two ways:

$$\vec{n}(a_2) \cdot (\vec{x} - \vec{x}_0) + a_1 = 0 \quad (4)$$

$$\left[\vec{n}_0(\theta) \cdot (\vec{x} - \vec{x}_0)\right] + \left[\vec{t}_0(\theta) \cdot (\vec{x} - \vec{x}_0)\right] a_2 + a_1 = 0. \quad (5)$$

Here, a_1 is again the normal displacement of the line from $\vec{x}_0$ and a_2 characterizes the normal $\vec{n}$ to the line L. The duality is clear: line L is represented by (a_1, a_2) in the Hough space, while a point $\vec{x}$ is represented by a line in $\vec{a}$-space.

Check **plotCstLines.m** to see how the ranges on (a_1, a_2) are set.

Assignment

The important pieces in the code **lineTutorial.m**:

```
% Read an image
im = pgmRead('im9h.pgm');

% set up sigma, minStrength
[edgelIm nrmGradIm dirIm] = cannyEdgels(im, sigma, minStrength);

% extract information from a crop box
[p, theta] = getEdgelData(edgelIm, dirIm, cropBox);

% Plot the constraint lines in the coefficient space.
plotCstLines(p, theta, lbl, thetaDeg, x0, 1.5*radius, hLine);
```

Check the figures at the end of this handout. The constraint line drawing code, **plotCstLines.m**, is called with three different choices of **thetaDeg** $= 0, 45, 90$.

YOUR JOB

g2.vis> grep -n JOB *.m

```
lineTutorial.m:170:
%%% YOUR JOB is to replace the constraint line plot with
%%% a grey-level plot of the Hough space for parameters (a_1,a_2),
%%% and investigate this Hough space for different bin sizes.
%%% Use a simple line-drawing algorithm for accumulating the
%%% votes for each edgel (eg. use Google to search
%%% for "Line Drawing Algorithm").  The algorithm you use
%%% will not be marked for efficiency.
```

```
lineTutorial.m:179:
%%% YOUR JOB is to add the least squares fit to the code in
%%% lineFitDemoLS.m.  This involves some thinking and a
%%% 2-line modification to plotCstLines.m
%%% (Search lineFitDemoLS.m and plotCstLines.m for YOUR JOB.)

lineTutorial.m:189:
%%% YOUR JOB is to replace the Hough space computed in
%%% question 1 above with an image of the values of the
%%% objective function using the Cauchy estimator.

lineTutorial.m:194:
%%% YOUR JOB is to replace the least squares fit used in
%%% lineFitDemoLS in Question 2 with an M-estimator, and
%%% to estimate multiple lines.

lineTutorial.m:199:
%%% YOUR JOB is to try to find line segment endpoints
%%% for the lines estimated in question 4.


plotCstLines.m:62:
%%%%%%%%%%%%%%%% YOUR JOB %%%%%%%%%%%%%%%%%%%%%%%
Q = zeros(2,2);
L = zeros(2,1);

lineFitDemoLS.m:58:
%%%%%%%%%%%%%%%%%%%%%%%%%  YOUR JOB %%%%%%%%%%%%%%%%%%%%%%%%
%% Problem 1.  The least squares matrix equations should be set up
%% by this point so that the equations for a are:
%%   Q a = -L
%% This could be done in plotCstLines, for example, or in
%% your own M-file.
```

How do we detect corner features?

Why bother?

Seek stable features for the *correspondence* problem.

Structure

Characterize the structure of the gray levels using spatial image gradients. Treat gradients as samples from a 2-dimensional Gaussian distribution.

How?

Using image gradients = $[E_x E_y]^T$, construct the covariance matrix

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}. \tag{6}$$

C is symmetric, hence eigen decomposition leads to orthogonal eigenvectors U and real, non-negative eigenvalues $\vec{\lambda}$:

$$U^T C U = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}. \tag{7}$$

Interpret

Assume $\lambda_1 \geq \lambda_2$, show eigenvectors encode edge directions and eigenvalues encode edge strength.

Computational aside

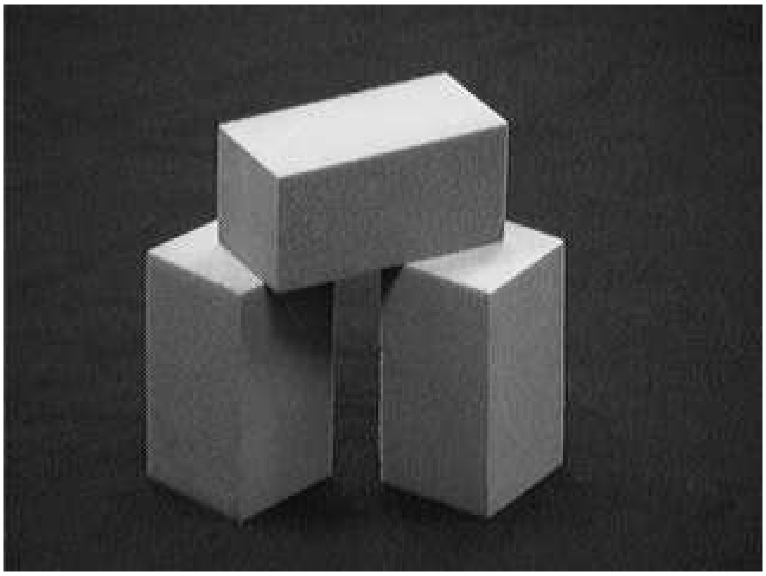
C is a 2×2 non-negative definite matrix. The eigen values are real and non-negative. To derive eigenvalues use the closed-form expression resulting from:

$$\det(C - \lambda I) = 0. \tag{8}$$

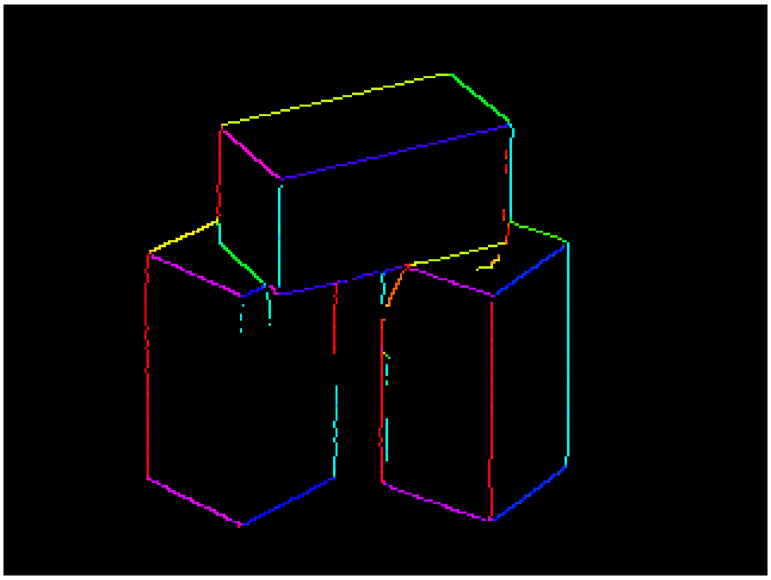
There is a similar closed form expression for the eigenvectors.

```
%-----------------------------------------------------------
% Blocks-World Example
%-----------------------------------------------------------
im = pgmRead('im9h.pgm');

%-----------------------------------------------------------
% Build Filter Kernels: Gaussian filter masks, along with
% derivatives.
%-----------------------------------------------------------
sigma = 2;
sigmaSqr = sigma*sigma;
gFiltSize = 2 * round(3.0 * sigma) + 1;
x = [1:gFiltSize] - round((gFiltSize+1)/2);
gFilt = exp(- x .* x / (2.0*sigmaSqr));
gFilt = gFilt/ sum(gFilt(:));
gxFilt = - (x / sigmaSqr) .* gFilt;
%gxxFilt = ((x / sigmaSqr).^2 - 1.0/sigmaSqr) .* gFilt;

%-----------------------------------------------------------
% Do separable convolutions
%-----------------------------------------------------------
%im = im/255;
dxIm = rconv2sep(im, gxFilt, gFilt);
dyIm = rconv2sep(im, gFilt, gxFilt);

%-----------------------------------------------------------
% Decide on the size of neighborhood
%-----------------------------------------------------------
nbr = 10;

% Pick a point
row = 106;  % (33581, 13347)
col = 63;

% crop the image and derivatives around (row,col)
```

```
smIm   = im(row-nbr:row+nbr,col-nbr:col+nbr);
smdxIm = dxIm(row-nbr:row+nbr,col-nbr:col+nbr);
smdyIm = dyIm(row-nbr:row+nbr,col-nbr:col+nbr);

% set up the covariance matrix
smEx2  = smdxIm.^2;
smEy2  = smdyIm.^2;
smExEy = smdxIm.*smdyIm;

C(1,1) = sum(sum(smEx2));
C(1,2) = sum(sum(smExEy));
C(2,1) = C(1,2);
C(2,2) = sum(sum(smEy2));
C11C22 = trace(C);

% compute eigenvalues
lambda1 = 0.5*(C11C22 + sqrt(C11C22.^2 - 4*(C(1,1)*C(2,2) - C(2,1)^2 )))
lambda2 = 0.5*(C11C22 - sqrt(C11C22.^2 - 4*(C(1,1)*C(2,2) - C(2,1)^2 )))
```



Range: [18, 238]
Dims: [240, 320]



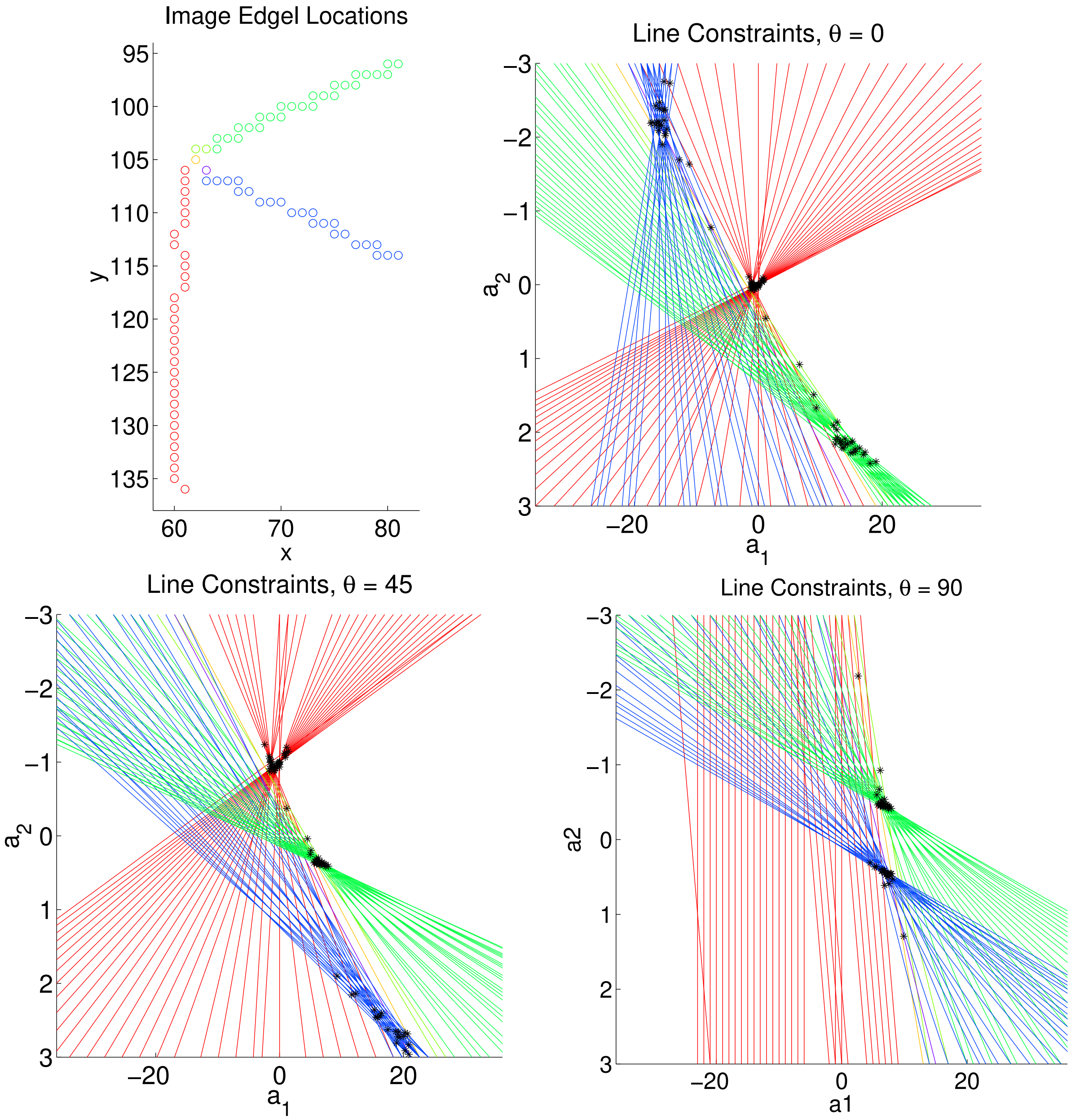
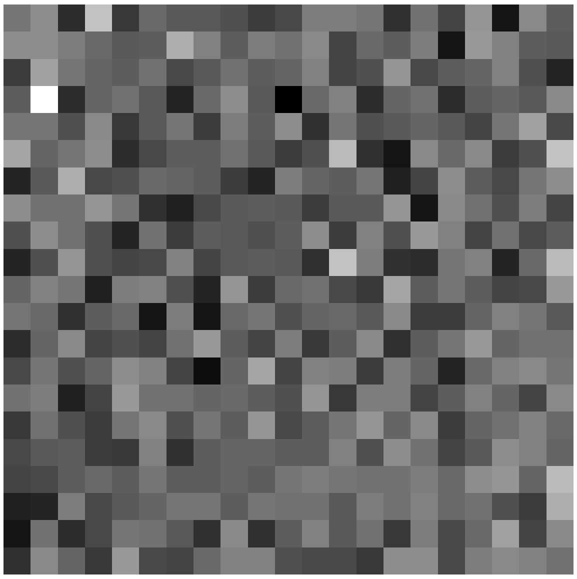
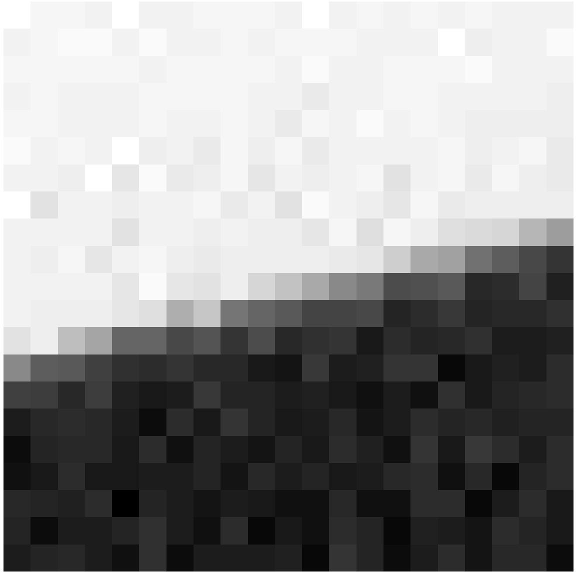


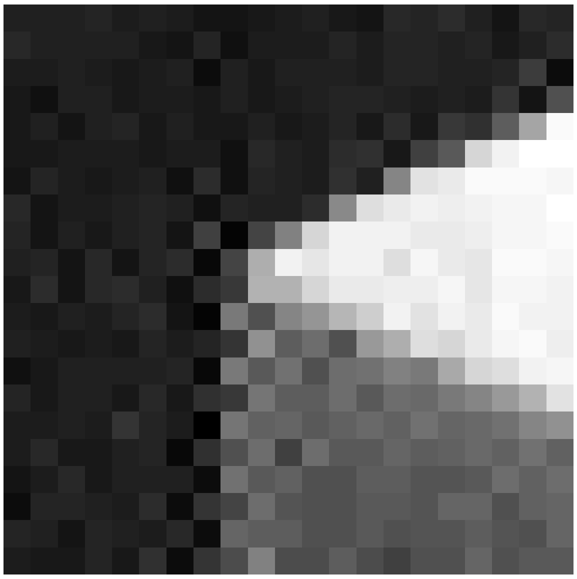
Image Edgel Locations
y
x
Line Constraints, θ = 0
a_2
a_1
Line Constraints, θ = 45
a_2
a_1
Line Constraints, θ = 90
a2
a1



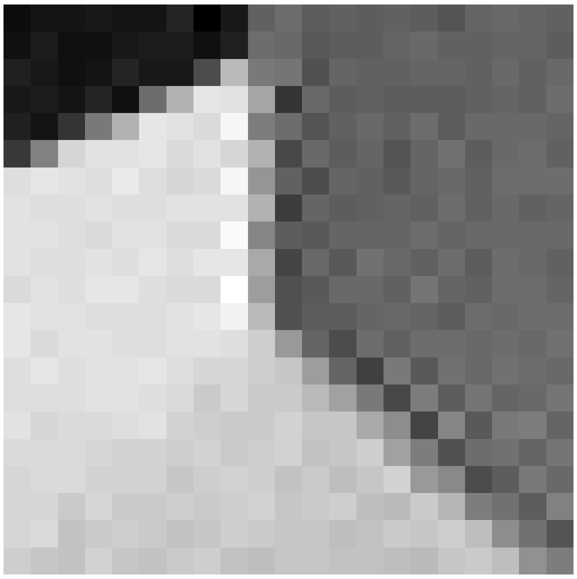
λ = (39, 28)



λ = (38476, 16)



λ = (33581, 13347)



λ = (25651, 14631)