corner detection;
hough transform
corner detection

what could we compute that would help detect corners?
edges

\[ I(x, y) \]

\[ \left( \frac{\partial I}{\partial x'}, \frac{\partial I}{\partial y} \right) \]
corners

\[ I(x, y) \]

\[ \left( \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) \]
corners

average \left( \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) ??
gradient covariance matrix

\[ C = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \]

summarizes second-order statistics of the gradient
quick digression

\[ Ax = \lambda x \]

• let \( A \) be a square \( N \times N \) matrix
• \( x \) is an eigenvector of \( A \)
• \( \lambda \) is its associated eigenvalue
• \( A \) can have at most \( N \) unique eigenvectors
• The rank(\( A \)) = number of eigenvectors with eigenvalues \( \neq 0 \).
• rank(\( A \)) = dimensionality of the range of \( A \)
case #1: uniform

\[
\left( \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) = (?, ?)
\]
case #2: single edge

\[
\left( \frac{\partial I}{\partial x'}, \frac{\partial I}{\partial y} \right) = (a, b)
\]
case #3: corner

\[
\begin{pmatrix}
\frac{\partial I}{\partial x'}, \frac{\partial I}{\partial y}
\end{pmatrix} = (a, b) \text{ and } (c, d)
\]
Tomasi-Kanade corner detector

• C has one large eigenvalue $\Rightarrow$ edge
• C has two large eigenvalues $\Rightarrow$ corner
implementation

• compute image gradient
• for each $M \times M$ neighborhood, compute $C$
• if smaller eigenvalue is larger than threshold record a corner (MATLAB: `eig`)
• nonmaximum suppression: keep strongest corner in each $M \times M$ window
corner detection

• application: good features for tracking, image correspondence, etc.

• why do corners make better features than edges?

• other corner detectors

• curvature in edge detector output

• color segmentation in neighborhoods

• others...
“good” image features (small neighborhoods of pixels)
camera calibration
3d scene reconstruction from image sequences
detecting lines

what is the difference between detecting edges and lines?
detecting lines

answer: local vs. global
brainstorm
the Hough transform

• “vote” for lines to which detected edges may belong
• votes will accumulate around actual lines
• issues:
  • what parameterization of lines?
  • how finely should we discretize line-space?
\[ y = ax + b \]
\[ y = ax + b \]
bucket size

- what resolution ("bucket size") in line space?
  - too small: poor performance on noisy data (aliasing)
  - too large: poor accuracy (false positives)
parameterization

• what’s wrong with slope-intercept?

\[ y = ax + b \]
parameterization

- what’s wrong with slope-intercept?
- non-uniform sampling of directions
- can’t represent vertical lines

\[ y = ax + b \]
parameterization

• better to use angle-distance parameterization
50% threshold  

70% threshold 

Image Credits: Bob Fisher
food for thought

• what does total accumulation of votes tell us about the line?

• what else can we detect with the Hough transform?

• circles? rectangles? human faces?
what is the dimensionality of parameter space?
what is the dimensionality of parameter space?

\[ 3 = 2 \text{ for center} + 1 \text{ for radius} \]

each pixel => 2D “sheet” of possible circles
what if we consider edge information?
often too noisy for final estimate, but good starting point for linear regression...
initial guess
(e.g., output of HT)

\[ y = mx + c \]
"best fit

\[ y = mx + c \]
least-squares fitting

\[ E = \sum ((mx_i + c) - y_i)^2 \]

putting the “squares” in least-squares
least-squares fitting

\[ y = mx + c \]

\[
\begin{pmatrix}
  x & 1 \\
 1 & c
\end{pmatrix}
\begin{pmatrix}
m \\
c
\end{pmatrix} = y
\]

\[ (0, 3), (2, 3), (4, 4), (-1, 2) \]

\[ A = \begin{pmatrix}
  0 & 1 \\
  2 & 1 \\
  4 & 1 \\
-1 & 1
\end{pmatrix}, \quad b = \begin{pmatrix}
  3 \\
  3 \\
  4 \\
  2
\end{pmatrix}, \quad p = \begin{pmatrix}
m \\
c
\end{pmatrix} \]

minimize: \((Ap - b)^T (Ap - b)\)
least-squares fitting

\[ E = (Ap - b)^T(Ap - b) \]

find \( p \) that minimizes:

\[ (Ap - b) = 0 \]

\[ Ap = b \]

\[ A^T Ap = A^T b \]

\[ p = (A^T A)^{-1} A^T b \]
outliers

- generative process: think of data as being generated by adding vertical Gaussian noise to best-fit line
- outliers = points with extremely low probability of occurrence within this model
- strongly influence least squares
robust estimation

• goal: develop parameter estimation methods insensitive to **small numbers of large errors**

• general approach: try to give large deviations less weight

• M-estimators: minimize some function other than \((y-f(x,a,b,...))^2\)
least absolute value fitting

• minimize \( \sum_i |y_i - f(x_i, a, b, ...)| \)

instead of \( \sum_i (y_i - f(x_i, a, b, ...))^2 \)

• points far away from trend get comparatively less influence
RANSAC

- **RAN**dom **SA**mple **C**onsensus: designed for bad data (in best case, up to 50% outliers)
- take many random subsets of data
  - compute least squares fit for each subset
  - see how many points agree: \((y_i-f(x_i))^2 < \text{threshold}\)
  - threshold user-specified or estimated from more trials
  - at end, use fit that agreed with most points
    - can perform one final LS with all inliers