Feature Detectors and Descriptors: Corners, Lines, etc.
Edges vs. Corners

- Edges = maxima in intensity gradient
Edges vs. Corners

- Corners = lots of variation in direction of gradient in a small neighborhood
Detecting Corners

• How to detect this variation?
• Not enough to check average $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$
Detecting Corners

• Claim: the following covariance matrix summarizes the statistics of the gradient

\[
C = \begin{bmatrix}
\sum f_x^2 & \sum f_x f_y \\
\sum f_x f_y & \sum f_y^2
\end{bmatrix}
\]

\[
f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y}
\]

Summations over local neighborhoods

– Can have spatially-varying weights (Gaussian, etc.)
Detecting Corners

- Examine behavior of C by testing its effect in simple cases
- Case #1: Single edge in local neighborhood
Case#1: Single Edge

• Let \((a,b)\) be gradient along edge
• Compute \(C \cdot (a,b)\):

\[
C \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \left[ \begin{array}{cc} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{array} \right] \begin{bmatrix} a \\ b \end{bmatrix} \\
= \sum (\nabla f)(\nabla f)^T \begin{bmatrix} a \\ b \end{bmatrix} \\
= \sum (\nabla f) \left( \nabla f \cdot \begin{bmatrix} a \\ b \end{bmatrix} \right)
\]
Case #1: Single Edge

- However, in this simple case, the only nonzero terms are those where $\nabla f = (a,b)$
- So, $C \cdot (a,b)$ is just some multiple of $(a,b)$
Case #2: Corner

- Assume there is a corner, with perpendicular gradients \((a,b)\) and \((c,d)\)
Case #2: Corner

• What is $C \cdot (a,b)$?
  – Since $(a,b) \cdot (c,d) = 0$, the only nonzero terms are those where $\nabla f = (a,b)$
  – So, $C \cdot (a,b)$ is again just a multiple of $(a,b)$

• What is $C \cdot (c,d)$?
  – Since $(a,b) \cdot (c,d) = 0$, the only nonzero terms are those where $\nabla f = (c,d)$
  – So, $C \cdot (c,d)$ is a multiple of $(c,d)$
Corner Detection

- Matrix times vector = multiple of vector
- Eigenvectors and eigenvalues!
- In particular, if $C$ has one large eigenvalue, there’s an edge
- If $C$ has two large eigenvalues, have corner
- Tomasi-Kanade corner detector
  - Variants you may hear about: Förstner 1986, Harris & Stephens 1988, Shi & Tomasi 1994
Corner Detection Implementation

1. Compute image gradient

2. For each $m \times m$ neighborhood, compute matrix $C$ (optionally using weighted sum)

3. If smaller eigenvalue $\lambda_2$ is larger than threshold $\tau$, record a corner

4. Nonmaximum suppression: only keep strongest corner in each $m \times m$ window
Corner Detection Results

- Checkerboard with noise

Trucco & Verri
Corner Detection Results
Corner Detection Results

Histogram of $\lambda_2$ (smaller eigenvalue)
Corner Detection

- **Application:** good features for tracking, correspondence, etc.
  - Why are corners better than edges for tracking?
- **Other corner detectors**
  - Look for curvature in edge detector output
  - Perform color segmentation on neighborhoods
  - Others...
Invariance

• Suppose you rotate the image by some angle
  – Will you still find the same corners?

• What if you change the brightness?

• Scale?
Key idea: compute some function $f$ over different scales, find extremum

- Common definition of $f$: LoG or DoG
- Find local minima or maxima over position and scale
Automatic scale selection

Lindeberg et al., 1996
Automatic scale selection
Automatic scale selection
Automatic scale selection
Automatic scale selection
Automatic scale selection
Automatic scale selection

\[ f(I_{i-1}^{m}(x,\sigma)) \]

\[ f(I_{i-1}^{m}(x',\sigma')) \]
Automatic scale selection

Normalize: rescale to fixed size
Fitting and Matching

• We’ve seen low-level detectors

• Next step: using output for higher-level tasks
  – Detection/fitting of more complex primitives
  – Matching
Detecting Lines

• What is the difference between line detection and edge detection?
  – Edges = local
  – Lines = nonlocal

• Line detection usually performed on the output of an edge detector
Detecting Lines

- Possible approaches:
  - Brute force: enumerate all lines, check if present
  - Hough transform: vote for lines to which detected edges might belong
  - Fitting: given guess for approximate location, refine it

- Second method *efficient* for finding unknown lines, but not always *accurate*
Hough Transform

• General idea: transform from image coordinates to parameter space of feature
  – Need parameterized model of features
  – For each pixel, determine all parameter values that might have given rise to that pixel; vote
  – At end, look for peaks in parameter space
Hough Transform for Lines

- Generic line: $y = ax + b$
- Parameters: $a$ and $b$
Hough Transform for Lines

1. Initialize table of buckets, indexed by $a$ and $b$, to zero

2. For each detected edge pixel $(x,y)$:
   a. Determine all $(a,b)$ such that $y = ax + b$
   b. Increment bucket $(a,b)$

3. Buckets with many votes indicate probable lines
Hough Transform for Lines
Hough Transform for Lines
Bucket Selection

• How to select bucket size?
  – Too small: poor performance on noisy data
  – Too large: poor accuracy, long running times, possibility of false positives

• Large buckets + verification and refinement
  – Problems distinguishing nearby lines

• Be smarter at selecting buckets
  – Use gradient information to select subset of buckets
  – More sensitive to noise
Difficulties with Hough Transform for Lines

• Slope / intercept parameterization not ideal
  – Non-uniform sampling of directions
  – Can’t represent vertical lines

• Angle / distance parameterization
  – Line represented as \((r, \theta)\) where
    \[x \cos \theta + y \sin \theta = r\]
Angle / Distance Parameterization

- Advantage: uniform parameterization of directions
- Disadvantage: space of all lines passing through a point becomes a sinusoid in \((r, \theta)\) space
Hough Transform Results
Hough Transform Results
Hough Transform

• What else can be detected using Hough transform?
• Anything, but *dimensionality* is key
Hough Transform for Circles

- Space of circles has a 3-dimensional parameter space: position (2-d) and radius
- So, each pixel gives rise to 2-d sheet of values in 3-d space
Hough Transform for Circles

• In many cases, can simplify problem by using more information
• Example: using gradient information

• Still need 3-d bucket space, but each pixel only votes for 1-d subset
Hough Transform for Circles – Secants

• 2-D bucket space: vote only for center, not $r$
Simplifying Hough Transforms

• Another trick: use prior information
  – For example, if looking for circles of a particular size, reduce votes even further
Fitting

• Output of Hough transform often not accurate enough
• Use as initial guess for fitting
Fitting Lines

Initial guess
Fitting Lines

Least-squares minimization
Fitting Lines

- As before, have to be careful about parameterization
- Simplest line fitting formulas minimize vertical (not perpendicular) point-to-line distance
- Closed-form solution for point-to-line distance, not necessarily true for other curves
Total Least Squares

1. Translate center of mass to origin
Total Least Squares

2. Compute covariance matrix, find eigenvector w. largest eigenvalue
Outliers

- Least squares assumes Gaussian errors
- **Outliers**: points with extremely low probability of occurrence (according to Gaussian statistics)
  - Can be result of *data association* problems
- Can have strong influence on 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 \[ \sum_i |y_i - f(x_i, a, b, \ldots)| \]

  instead of \[ \sum_i (y_i - f(x_i, a, b, \ldots))^2 \]

- Points far away from trend get comparatively less influence
Example: Constant

- For constant function \( y = a \),
  minimizing \( \sum (y-a)^2 \) gives \( a = \text{mean} \)
- Minimizing \( \sum |y-a| \) gives \( a = \text{median} \)
Doing Robust Fitting

- In general case, nasty function: discontinuous derivative
- Numerical methods (e.g. Nelder-Mead simplex) sometimes work
Iteratively Reweighted Least Squares

- Sometimes-used approximation: convert to iterated *weighted* least squares

\[ \sum_{i} |y_i - f(x_i, a, b, \ldots)| \]

\[ = \sum_{i} \frac{1}{|y_i - f(x_i, a, b, \ldots)|} (y_i - f(x_i, a, b, \ldots))^2 \]

\[ = \sum_{i} w_i (y_i - f(x_i, a, b, \ldots))^2 \]

with \( w_i \) based on previous iteration
Iteratively Reweighted Least Squares

- Different options for weights
  - Avoid problems with infinities
  - Give even less weight to outliers

\[ w_i = \frac{1}{|y_i - f(x_i, a, b, \ldots)|} \]

\[ w_i = \frac{1}{k + |y_i - f(x_i, a, b, \ldots)|} \]

\[ w_i = \frac{1}{k + (y_i - f(x_i, a, b, \ldots))^2} \]

\[ w_i = e^{-k(y_i - f(x_i, a, b, \ldots))^2} \]
Outlier Detection and Rejection

• Special case of IRWLS: set weight = 0 if outlier, 1 otherwise

• Detecting outliers: \((y_i-f(x_i))^2 > \text{threshold}\)
  – One choice: multiple of mean squared difference
  – Better choice: multiple of median squared difference
  – Can iterate…
  – As before, not guaranteed to do anything reasonable, tends to work OK if only a few outliers
RANSAC

- **RANdom SAmple Consensus**: designed for bad data (in best case, up to 50% outliers)

- Take many *minimal* random subsets of data
  - Compute least squares fit for each sample
  - 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 do one final least squares with all inliers
Feature Descriptors

- Feature matching useful for:
  Image alignment (e.g., mosaics), 3D reconstruction, motion tracking, object recognition, indexing and database retrieval, robot navigation, etc.
Properties of Feature Descriptors

- Easily computed
- Easily compared (compact, fixed-dimensional)
- Invariant
  - Translation
  - Rotation
  - Scale
  - Change in image brightness
  - Change in perspective?
Rotation Invariance for Feature Descriptors

- Rotate window according to dominant orientation
  - Eigenvector of C corresponding to maximum eigenvalue
**Scale Invariant Feature Transform**

- Take 16×16 window around detected feature
- Create histogram of thresholded edge orientations
Full SIFT Descriptor

- Divide 16×16 window into 4×4 grid of cells
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128-dimensional descriptor
Properties of SIFT

- Fast (real-time) and robust descriptor for matching
  - Handles changes in viewpoint ($\sim 60^\circ$ out of plane rotation)
  - Handles significant changes in illumination
  - Lots of code available