Multi-stable Perception

Necker Cube
Spinning dancer illusion, Nobuyuki Kayahara
Feature Matching and Robust Fitting

Read Szeliski 7.4.2 and 2.1

Computer Vision

James Hays
**Project 2**

Overview
The goal of this assignment is to create a local feature matching algorithm using techniques described in Section 7.2.2. The pipeline requires a detection stage, followed by a feature extraction stage, and then matching between the two stages.

**Brief**
- Due: Check Grades for up-to-date information.
- Project materials including report template: Project 2.
- Baseline through crowdsourcing.
- Key files: `your_user_name/report_type/report_name_proj2.pdf`

---

**Algorithm 1: Harris Corner Detector**

Compute the horizontal and vertical derivatives $L_x$ and $L_y$ of the image by convolving the original image with a Sobel filter.

1. Compute the Harris matrix $M$ from the outer products of these gradients. (The matrix $A$ is $M$.)

2. For each image point $(x, y)$, compute the Harris corner score using the formula:

   $$G(x, y) = \text{det}(M) - k \cdot \text{tr}(M)^2$$

   Where $k$ is a constant, commonly set to 0.04.

3. Detect corners by finding local maxima of $G$.

**Figure 1**

The top 100 most salient local feature matches from a baseline implementation of project 2. In this case, 66 were correct (lines shown in green), and 11 were incorrect (lines shown in red).

---

Note: The slides given in this report should be used as a template to create your own presentation. If you are given new credit implications by your instructor, please follow those instructions. Otherwise, the slides should be modified to fit your own project. Sufficient credit will be awarded for your extra credit implementations.

---

SIFT Local Feature Matching

- Algorithm 1: Harris Corner Detector
- Algorithm 2: Feature Description
- Algorithm 3: Feature Matching

**Figure 2**

Feature matching results using SIFT. The top 100 matches are shown, with green lines indicating correct matches and red lines indicating incorrect matches.

---

Evaluation of the code:
- The code implemented in the starter code includes the visualizations to help you understand the process.
- The starter code also includes a function to evaluate the code.
- The code is designed to be simple and easy to understand.

---

Functions

- `match_features.py`: Takes two images as input and outputs the matched features.
- `describe_features.py`: Takes a feature descriptor and outputs a set of descriptors.
- `match_points.py`: Takes two sets of descriptors and outputs the matched points.

---

The code provided in the starter code includes the visualizations to help you understand the process. The code is designed to be simple and easy to understand. You should use these functions to evaluate the code and make adjustments as needed.
This section: correspondence and alignment

• Correspondence: matching points, patches, edges, or regions across images
Review: Local Descriptors

• Most features can be thought of as templates, histograms (counts), or combinations

• The ideal descriptor should be
  – Robust and Distinctive
  – Compact and Efficient

• Most available descriptors focus on edge/gradient information
  – Capture texture information
  – Color rarely used
Can we refine this further?
Fitting: find the parameters of a model that best fit the data

Alignment: find the parameters of the transformation that best align matched points
Fitting and Alignment

• Design challenges
  – Design a suitable **goodness of fit** measure
    • Similarity should reflect application goals
    • Encode robustness to outliers and noise
  – Design an **optimization** method
    • Avoid local optima
    • Find best parameters quickly
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Simple example: Fitting a line
Least squares line fitting

- **Data**: \((x_1, y_1), \ldots, (x_n, y_n)\)
- **Line equation**: \(y_i = mx_i + b\)
- **Find** \((m, b)\) to minimize

\[
E = \sum_{i=1}^{n} (y_i - mx_i - b)^2
\]

\[
E = \sum_{i=1}^{n} \left( \begin{bmatrix} x_i \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|Ap - y\|^2
\]

\[
\frac{dE}{dp} = 2A^TAp - 2A^Ty = 0
\]

\[
A^TAp = A^Ty \Rightarrow p = \left(A^TA\right)^{-1}A^Ty
\]

**Matlab**: \(p = A \backslash y;\)

**Python**: \(p = \text{numpy.linalg.lstsq}(A, y)\)

Modified from S. Lazebnik
Least squares (global) optimization

Good
• Clearly specified objective
• Optimization is easy

Bad
• May not be what you want to optimize
• Sensitive to outliers
  – Bad matches, extra points
• Doesn’t allow you to get multiple good fits
  – Detecting multiple objects, lines, etc.
Least squares: Robustness to noise

- Least squares fit to the red points:
Least squares: Robustness to noise

- Least squares fit with an outlier:

Problem: squared error heavily penalizes outliers
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Robust least squares (to deal with outliers)

General approach:

\[
\text{minimize} \quad \sum_i \rho(u_i(x_i, \theta); \sigma)
\]

\[
u^2 = \sum_{i=1}^n (y_i - mx_i - b)^2
\]

\(u_i(x_i, \theta)\) – residual of \(i^{th}\) point w.r.t. model parameters \(\theta\)

\(\rho\) – robust function with scale parameter \(\sigma\)

The robust function \(\rho\)

- Favors a configuration with small residuals
- Constant penalty for large residuals
Choosing the scale: Just right

The effect of the outlier is minimized
The error value is almost the same for every point and the fit is very poor

Choosing the scale: Too small
Choosing the scale: Too large

Behaves much the same as least squares
Robust estimation: Details

• Robust fitting is a nonlinear optimization problem that must be solved iteratively
• Least squares solution can be used for initialization
• Scale of robust function should be chosen adaptively based on median residual
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Other ways to search for parameters (for when no closed form solution exists)

• Line search (see also “coordinate descent”)
  1. For each parameter, step through values and choose value that gives best fit
  2. Repeat (1) until no parameter changes

• Grid search
  1. Propose several sets of parameters, evenly sampled in the joint set
  2. Choose best (or top few) and sample joint parameters around the current best; repeat

• Gradient descent
  1. Provide initial position (e.g., random)
  2. Locally search for better parameters by following gradient
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Fitting and Alignment: Methods

• Global optimization / Search for parameters
  – Least squares fit
  – Robust least squares
  – Other parameter search methods

• Hypothesize and test
  – Generalized Hough transform
  – RANSAC
Hough Transform: Outline

1. Create a grid of parameter values

2. Each point votes for a set of parameters, incrementing those values in grid

3. Find maximum or local maxima in grid
Hough transform


Given a set of points, find the curve or line that explains the data points best

\[ y = m x + b \]

Hough space

Slide from S. Savarese
Hough transform

Slide from S. Savarese
Hough transform


Issue: parameter space \([m,b]\) is unbounded…
Hough transform


Issue: parameter space \([m, b]\) is unbounded...

Use a polar representation for the parameter space

\[
x \cos \theta + y \sin \theta = \rho
\]
Hough transform - experiments

Slide from S. Savarese
Hough transform - experiments

Need to adjust grid size or smooth

Slide from S. Savarese
Hough transform - experiments

Issue: spurious peaks due to uniform noise
1. Image → Canny Edge Detection
2. Canny $\rightarrow$ Hough votes
3. Hough votes $\rightarrow$ Edges

Find peaks and post-process
Hough transform example
Finding lines using Hough transform

• Using m,b parameterization
• Using r, theta parameterization
  – Using oriented gradients
• Practical considerations
  – Bin size
  – Smoothing
  – Finding multiple lines
  – Finding line segments
Hough Transform

- How would we find circles?
  - Of fixed radius
  - Of unknown radius
  - Of unknown radius but with known edge orientation
Hough transform for circles

- Grid search equivalent procedure: for each \((x,y,r)\), draw the corresponding circle in the image and compute its “support”
Hough Transform

• How would we find circles?
  – Of fixed radius
  – Of unknown radius
  – Of unknown radius but with known edge orientation
Hough transform for circles

Image space

Hough parameter space

\[(x, y) + r \nabla I(x, y)\]

\[(x, y) - r \nabla I(x, y)\]
Hough transform conclusions

Good
• Robust to outliers: each point votes separately
• Fairly efficient (often faster than trying all sets of parameters)
• Provides multiple good fits

Bad
• Some sensitivity to noise
• Bin size trades off between noise tolerance, precision, and speed/memory
  – Can be hard to find sweet spot
• Not suitable for more than a few parameters
  – Grid size grows exponentially

Common applications
• Line fitting (also circles, ellipses, etc.)
• Object instance recognition (parameters are affine transform)
• Object category recognition (parameters are position/scale)