Targil 11:
Robust Techniques
RANSAC
Hough Transform

What are robust methods

Goal of many algorithms: extract parametric information from an image (parametric fitting)
- Extracting edges
- Extracting motion model from correspondence points

In real images: inaccurate data
- Occlusions
- Noise
- Wrong feature extraction
- Data is likely to include multiple models / structures

Goal of robust method: tolerate outliers points

Example: Robust Computation of 2D Homography
from Hartley & Zisserman

Simple Model Fit Problem
Estimate a straight line fit to a set of 2D points

Least Square (LS) error is not robust: one outlier point may cause a very big mistake
Another outlier, another disaster

Close-up of the poor fit to the “true” data.

Robust Norm

A quadratic $\rho$ function gives too much weight to outliers. Instead, use robust norm:

$$\rho(r, \sigma) = \frac{r^2}{\sigma^2 + r^2}$$

We need to set $\sigma$

Parameters:

- unknown scale

Robust Norm

The impact of tuning $\sigma$

Just right

Too small

Too large

Simple Model Fit Problem

Estimate a straight line fit to a set of 2D points

How can we do better?

Assumption: Given the correct model, we can validate this model

RANSAC (RANdom SAmple Consensus)

The idea: Randomly choose points and check the support of model resulted from those points – choose model with max support

Support of the line: the number of points that lie within a distance threshold
**RANSAC – The Algorithm**

Iterate N times:
- Randomly Choose S points from the data
- Solve the problem using the S points
- Check the solution using the rest of the points (the support)

At the end:
Use all the points which “agree” to compute the exact model!

---

**Sufficient Number of Iterations (N)**

\[ p=0.99 \]
\[ w : \text{inliers probability,} \]
\[ e = 1 - w : \text{outlier probability} \]

<table>
<thead>
<tr>
<th>Sample size</th>
<th>2%</th>
<th>5%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>19</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>24</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>26</td>
<td>57</td>
<td>146</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>24</td>
<td>37</td>
<td>97</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>54</td>
<td>163</td>
<td>588</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>44</td>
<td>78</td>
<td>272</td>
<td>1177</td>
</tr>
</tbody>
</table>

---

**Adaptively determining the number of Iterations**

- \( e \) is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield \( e=0.2 \)

- \( N=\infty, \text{ sample_count}=0 \)
- While \( N>\text{sample_count} \) repeat
  - Choose a sample and count the number of inliers
  - Set \( e=1-(\text{number of inliers})/(\text{total number of points}) \)
  - Recompute \( N \) from \( e \)
  - Increment the \( \text{sample_count} \) by 1
- Terminate

---

**RANSAC discussion**

**Advantages:**
- General method suited for a wide range of model fitting problems
- Easy to implement and easy to calculate its failure rate

**Disadvantages:**
- Only handles a moderate percentage of outliers without cost blowing up
- Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)

---

**Hough Transform**

(proposed by Hough in ~1959)

Finding Shapes
- Straight Lines
- Rectangles
- Circles
- Others

More generally:
A parametric voting
Applications

- Find signs in the images (rectangles, circles).
- 3D reconstruction of buildings (lines).
- Find the eye in a face image.
- Tracking
- More...

The challenge:
- The image contains a lot of irrelevant information.
- There are partial occlusions.
- The image might be noisy

Edge Detection Review

1) Noise Smoothing

2) Edge Enhancement

Vertical

\[-1 0 1\]

Horizontal

\[-1 0 1\]

3) Threshold

How Do we detect lines from Edge Images

- Given an edge image, we would like to group sets of edge pixels to identify dominant lines in an image.
- One possibility is template matching in the edge image
  - Generate hypothetical line “kernel” and convolve with the image
  - Requires a large set of masks to accurately localize lines, which is computationally expensive
  - We will instead perform an image transformation where edge pixels vote for lines
- Eventually reduces the problem to vote thresholding

How Do we detect lines from Edge Images

The Parameters Space

The line equation: \( y = ax + b \) (slope-intercept representation)

Each line can be represented with 2 parameters: \((a,b)\)
If we define the a parapets space, each line is a point in this space.

For each point on this line \((x,y)\), there are infinite number of lines that goes through it in the parameters space:

\( b = x 
\)
The Duality Property
A point is mapped to a line, and a line is mapped to a point.

Example: Slope-Intercept Representation

Issues to think of...
- **Representation**: Vertical lines are not covered by the representation \( y = ax + b \).
- **Range**: Selecting the range of values for the table (\( a \) and \( b \) are not bounded!)
- **Quantization**: choosing a grid.
- **Thresholding**: Similar lines will get similar scores.

A better representation
Use a polar representation instead:

\[
d = x \cdot \cos(\theta) + y \cdot \sin(\theta)
\]

In the polar representation...

Hough Transform: use a voting scheme
- Apply an edge detector on the image (gradient + threshold, or using Canny’s method)
- Prepare a table \( h(a, b) \) for \( a, b \) in a certain range.
- Loop over the image for each pixel \((x, y)\) on an edge, vote for the cells \((a, b)\) which satisfy the equation \( y = ax + b \). (increase the cell accumulator \( h(a, b)++ \))
- Choose the cell with the maximal value (or the cells which pass a given threshold)
Another Example

Original image + edges
Hough (before threshold)

Image + detected lines

Range and Quantization

In the polar representation it is easier to determine the range:
- The values of d and of $\theta$ are now bounded!
- The resolution can be application dependent. We have natural units (pixels for d, and degrees for $\theta$)

Quantization:
- Option1: Vote for the "nearest" cell (i.e - for each $\theta$ take the nearest d).
- Option2: Perform a weighted vote.

Smoothing

Without smoothing

With smoothing

Making the Decision

Q. How can we find the two most dominant lines in the image?

The problem: if $(r, \theta)$ is the cell with the maximal value, then $(r+\epsilon, \theta+\epsilon)$ will get a high score too.

Solution 1: Search only for local maxima.

Solution 2: "Back Mapping" - Perform a second vote, where each point votes only for the best line passing through it.

Making the Decision (cont')

Using more image features.
Example – using the local gradient
Detecting Complex Shapes – Circles

Naive solution:
- Construct a table \( h(a,b,r) \).
- For each pixel \((x,y)\) which is edge in the image, vote for all the cells satisfying:

\[
(x - a)^2 + (y - b)^2 = r^2
\]

Problem:
- The size of the table is \( O(N^3) \).
- The voting for each pixels takes: \( O(N^2) \).

Solution: Use the gradient information - (Solve for the center \((a,b)\) and then for \(r\))

Circle Example

With no orientation, each token (point) votes for all possible circles. With orientation, each token can vote for a smaller number of circles.

Finding Coins

Finding Coins (Continued)

Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

Coin finding sample images from: Vivik Kwatra
Another Example

How can we find the “dominant” direction of the lines?