Issues in region analysis

First, consider an example quality control problem

- Find all buttons in the image
- Classify each button as either “good” (free of defects) or “bad” (defective)
- How should you proceed?

- The final system should be able to repeat the analysis for tens (or hundreds) of images every minute!
Outline of a possible approach

• Acquire a gray-scale image
  – Each pixel represents a brightness value
• Threshold the image
  – Now each pixel value represents “foreground” or “background”
• Identify individual regions
  – Assign a unique label to each connected set of foreground pixels
  – This is called “region labeling”

• . . . (see next page)

• . . .
• Clean up the image
  – Remove noise (= small regions or small holes in regions)
  – Possibly modify region shape, etc.
• Extract features for each region
  – Example features: width, height, perimeter, number of holes, orientation, etc.
• Classify each region
  – “Good”
  – “Bad”
  – “Unknown”

5554: Packet 5
Major issues in dealing with image regions

- Region detection
  - thresholding, etc.
  - region labeling
- Region representation
- Region refinement
  - noise reduction
  - shape modification
- Region analysis
  - region properties
  - feature extraction

Closely related topics:
- 2D shapes
  - representations
  - properties
- 2D shape boundaries
  - representations
  - properties

Binary image representation

- A region detection process (such as thresholding) typically results in a binary image
  (each pixel is either “foreground” or “background”)
- The usual image representation is a rectangular array, sometimes called a “bit map” or a “spatial occupancy array”
- For example, the image at the right could be represented as follows:

```
0000000000
0010000000
0010000000
0000100000
0000010000
0000001000
0000000000
0000000000
```
• We can also describe a region (or an entire binary image) by **exhaustive enumeration**:  

List the image coordinates of all relevant pixels

• For example, we can use the set $S$ to refer to all of the foreground pixels for the image at the right:  

$$S = \{(1, 2), (2, 2), (3, 5), (4, 6)\}$$

• This uses *(row, column)* coordinates, with the upper left pixel (0,0)

• We can also describe a region (or an entire binary image) using **run-length encoding**

• A “run” is a sequence of pixels having the same value

• Typically we consider a single image row at a time

• One convention is to give the number of 1s first, for each row; in this case, the row above would be represented as $|5, 3, 1, 1, 4|$
• Represent the image below using run-length encoding

More examples

111 11 11
111 11 11
111 11 11
111 11 11
111 11 11
Fundamental concepts (con't)

• A region is a connected portion of an image

(We often consider an image to be partitioned into nonoverlapping regions)

• Now, what exactly does it mean to be “connected”?

• Does this image contain 1 or 2 foreground regions?

Pixel neighbors

• Normally a digital image is composed of a rectangular grid of pixels
• Consider a pixel $p$ that is within the grid

The dark pixels above are called the 4-neighbors of $p$

The dark pixels above are called the 8-neighbors of $p$
• For a given pixel at \((row, column)\) location \((r, c)\),
the 4-neighbors are at locations
\((r-1, c), (r, c-1), (r, c+1), \text{ and } (r+1, c)\)

• For a given pixel \((r, c)\), the 8-neighbors are at locations
\((r-1, c-1), (r-1, c), (r-1, c+1), (r, c-1), (r, c+1), (r+1, c-1),
(r+1, c), \text{ and } (r+1, c+1)\)

Paths

• A path is a sequence of pixel locations,
such that each pixel in the sequence is a neighbor of the pixel before (or after) it
- If only 4-neighbors are used to describe a path, then this is sometimes called a 4-path.
- Example:

  ( (1, 2), (1, 3), (1, 4), (2, 4), (2, 5),
   (2, 6), (2, 7), (3, 7), (4, 7) )

- Similarly, an 8-path is a sequence of 8-neighbors.
- Example:

  ( (1, 2), (1, 3), (1, 4), (2, 5),
   (2, 6), (3, 7), (4, 7) )
A related topic:
chain code

- A **chain code** is a means of representing digital curves, such as region boundaries or other paths.
- The usual coding scheme is as follows:

```
  3  2  1
  4  P  0
  5  6  7
```

- From a given starting pixel on the curve, record the next pixel using the appropriate code number (and repeat).

**Example**

Chain code:
```
  3  2  1
  4  P  0
  5  6  7
```

Difference chain code:
```
  3  2  1
  4  P  0
 -3 -2 -1
```
Connectivity

• Let \( S \) represent a set of pixels (normally in the foreground)
• We say that 2 pixels \( p \) and \( q \) are connected in \( S \) if
  – there exists a path from \( p \) to \( q \), such that
  – every pixel in the path belongs to \( S \)
• (Either 4-neighbors or 8-neighbors can be used, and this should be specified)
• In the example at the right, we say that \( p \) and \( q \) are 8-connected, but they are not 4-connected

• A **connected component** of an image is a set of pixels such that every pixel of the set is connected to every other pixel of the set

• In the example at the right, if we assume 4-connectivity, then there are 2 (maximally) connected components of the foreground
• But if we assume 8-connectivity, then there is 1 (maximally) connected component
• Consider this small binary image:

• For the foreground, if we decide to use 8-connectivity, then just 1 connected component is present.
• But what about the background? If we also use 8-connectivity for the background, then we seem to have a problem.

(Should there be 2 background regions, or only 1?)

• Other problems become apparent with this example, if the same type of connectivity is used for both foreground and background.

• The solution:
use one type of connectivity for the foreground, and use the other type of connectivity for the background!
Counting the number of regions

- **Goal:**
  find the number of foreground regions in an image

- **Simplifying assumptions:**
  - Detect 4-connected regions
  - Each region is “simply connected” (no holes are present in any region)

- Because of the rectangular pixel grid, a fast method is possible

Consider some possible region shapes

<table>
<thead>
<tr>
<th>E</th>
<th>I</th>
<th>(\Delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- \(E\) = number of “external” corners
- \(I\) = number of “internal” corners
- \(\Delta = E - I\)

- Under our simplifying assumptions, the number of regions is \((E - I) / 4\)
- This treats each pixel as a small rectangular region or sub-region
- It is possible to compute \(E\) and \(I\) quickly

---

**Counting regions**
(Adapted from Shapiro and Stockman, p. 68)

```plaintext
//Compute the number of 4-connected regions in binary image B.
//The regions are assumed to contain no holes.

procedure count_objects(B)
{
  E := 0     // number of "external" corners
  I := 0     // number of "internal" corners
  for all rows in image B
    for all columns in image B
      {
        if an external corner is present at the current pixel then
          E := E + 1
        if an internal corner is present at the current pixel then
          I := I + 1
      }
  return ((E - I) / 4)
}
```
Looking for corners

- A typical external corner corresponds to 1 of these patterns:

```
  0  1
  1  0
```

- A typical internal corner corresponds to 1 of these patterns:

```
  0  1
  1  0
```

- Assume that the corner location for each pattern is at the center of the $2 \times 2$ grid
- Each of these patterns is called a **mask** or a **template**
- The idea is to scan the image in raster order, searching for matches to any of these patterns

Looking for corners

- Shapiro and Stockman make 2 more assumptions to simplify processing
  - The outermost rows and columns contain only background pixels (sometimes this is called a "guard band")
  - The image contains no cases like these:

```
  0  1
  1  0
```

- Now, for a given pixel location $(r, c)$, we imagine a "window" that allows us to look at a $2 \times 2$ neighborhood
- In other words, consider the 4 pixels at locations $(r, c)$, $(r, c+1)$, $(r+1, c)$, and $(r+1, c+1)$
- If there are 3 foreground pixels in the window, then an internal corner is present (procedure `internal_match")
- If there are 3 background pixels in the window, then an external corner is present (procedure "external_match")
Counting regions: more details

//Compute the number of 4-connected regions in binary image B.  
//Image B is of size ROWS × COLUMNS. 
//The regions are assumed to contain no holes.

procedure count_objects(B)
{
  E := 0   // number of "external" corners
  I := 0   // number of "internal" corners
  for  r := 0 to ROWS-2
    for  c := 0 to COLUMNS-2
    {
        if external_match(r, c) then 
            E := E + 1
        if internal_match (r, c) then 
            I := I + 1
    }
  return ((E – I) / 4)
}

---

External corners:

0 0 0 0
0 1 0 1
0 0 0 1
0 0 0 0

Internal corners:

0 0 0 0
0 1 0 1
0 1 1 0
0 0 0 0
Larger example

e = 7
i = 3
(e-i)/2 = 1

Region labeling
(also called connected component labeling)

- Goal: Assign a unique label to each region of an image

Example:

- For simplicity, let’s first assume that we want to identify pixels belonging to separate 4-connected regions
- Also assume that the input image contains pixel values of only 1 (foreground) and 0 (background)
- We want to change each 1 to a unique ‘label’ – just use numbers > 1
Region labeling

// Compute the number of regions in binary image B.

procedure label_objects(B)
{
    for all rows in image B
        for all columns in image B
            {
                if the current pixel is foreground and has not been labeled then
                    {
                        select a new label
                        assign the new label to the current pixel
                        find and assign the new label to all foreground pixels
                            that are connected to the current pixel
                    }
            }
}