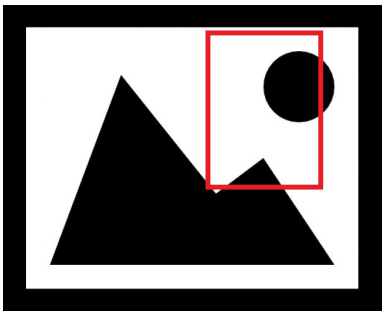
Pattern Finder

Christophe Saad

October 2017



1 Description

PatternFinder is a Java application which matches an image in a background. It can use two techniques for pattern matching :

1. Distance Based Search
2. Similarity Based Search

Both techniques operates the same way: we slide the pattern along the background, calculate some coefficient which will help us decide of the position of the pattern.

2 Content

1. PatternFinder project folder
2. PatternFinder runnable jar file
3. README.pdf

3 Structure

3.1 Image Processing

We begin by developping tools for processing images. Transforming an image into a table of RGB (int[][]) and then be able to extract each component of this RGB color (done by manipulating bits, shifting, and, or...). We also need to transform each RGB color into a shade of gray and vice versa.

3.2 Distance Based Search

For each possible position of the pattern image in the background image, we calculate the distance between pixel values and put them in a matrix, the distance matrix. We use the absolute mean error to measure the distance between two pixels.

$$d(B,P) = \frac{\sum_{c\in\{R,G,B\}} |B_c - P_c|}{3}$$

And then we calculate the distance between the pattern image P and the sub-background image starting at row and col from the full background image B.

$$D(B, P, row, col) = \frac{\sum_{(i,j)\in dim(P)} d(B[row+i, col+j], P[i,j])}{\#pixels\ of\ B}$$

3.3 Similarity Based Search

Same as 3.2 but we use a coefficient of correlation instead to fill a similarity matrix.

$$C(B, P, row, col) = \frac{\sum_{(i,j)\in dim(P)} (B[row+i, col+j] - \tilde{B}) \times (P[i,j] - \tilde{P}))}{\sqrt{\sum_{(i,j)\in dim(P)} (B[row+i, col+j] - \tilde{B})^2 \times \sum_{(i,j)\in dim(P)} (P[i,j] - \tilde{P})^2}}$$

where $\tilde{P}$ and $\tilde{B}$ are the mean values of shades of gray of the images.

3.4 Finding best fit

Two methods to find the n best values:

1. Brute force : walk along the matrix and extract best value n times $\rightarrow O(n \times p)$
2. Quicksort : sort matrix elements using quicksort and then get the best values $\rightarrow O(plog(p))$

where p is the number of pixels of B