ECTE331: Real-time systems Project

Problem 1: Single Thread and Multi-Thread Implementation of Template Matching Algorithm

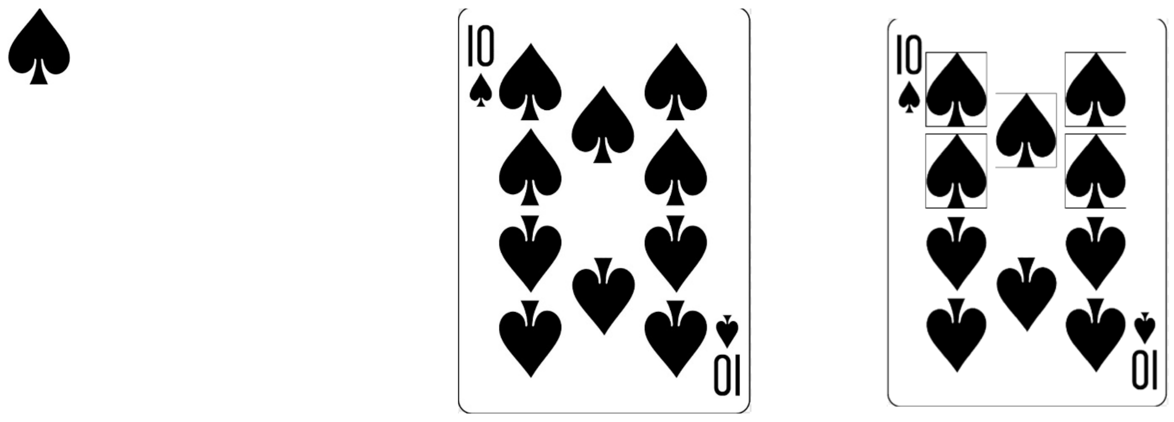
Description:

Template matching is an image processing technique for finding sections of a **source image** (`SourceImage`) matching a predefined **template image (**`TemplateImage`**).** Figure 1 gives an algorithm implementation of this technique tailored for the type of images required in this task. The steps of the algorithm are given in a Matlab-like pseudo code **(the first array index is 1 and not 0 as in Java)**

```
im1= read_image(TemplateImage);
im2=read_image(SourceImage);
% apply template matching
result=template_matching(im1,im2);
display im1, im2, result
End;

Function template_matching:
result=template_matching(TemplateImage, SourceImage):
[r1, c1]=size(SourceImage);
[r2, c2]=size(TemplateImage);
TempSize=r2*c2;
Minimum=1000000;
% Absolute difference Matrix for i=1:(r1-r2+1)
  for j=1:(c1-c2+1)
    Nimage=Target(i:i+r2-1,j:j+c2-1);
    absDiff=sum(sum(abs(Nimage-Template)))/TempSize;
    absDiffMat(i,j)=absDiff;

      if absDiff <Minimum
        Minimum=absDiff;
      end;
  end
end

ratio=10;  % for calibration
Threshold=ratio*Minimum;
% return the coordinates of all absDiffMat elements which are <= Threshold
coordinates= find(absDiffMat<=Threshold);

Write the result image by drawing a rectangle on the source image at each element of coordinates list.
```

Figure 1: Template Matching Algorithm

The algorithm consists of sliding `TemplateImage` across `SourceImage`, column by column through all the rows. At every new coordinate, the mean of the absolute difference between `TemplateImage` and the underlying section of `SourceImage` is calculated as given in Figure. 1. The coordinates for which the mean absolute difference value is below or equal `Threshold` correspond to the section of `SourceImage` which matches `TemplateImage`. Each resulting matched section is then delimited with a rectangle, see Figure 2.



(a) Template image (size and orientation do matter) (b) Source Image (c) Result Image

Figure 2: Template Matching Algorithm Execution.

Requirements:

Essentially, implement in JAVA the algorithm of Figure 1 using **single thread** and **multi-thread** implementation; then compare the respective ***execution times***[1] of the two implementations. Furthermore, the following requirements should be adhered to:

- Your code should make use of only the standard Java APIs; no other third-party API such OpenCV can be used.
- Your code should handle any image dimensions. However, the testing should be conducted using the provided images: Source: 'TenCardG.jpg' and Template: 'OneG.jpg'.
- For the multi-thread implementation of the algorithm, every thread should apply the algorithm on a subpart of `SourceImage`. The threads of your application should be defined in an array of length name: `numOfThreads`.
- Compare the execution times of the single thread and multi-thread implementations for different values of `numOfThreads`. The execution times should include the overhead of launching the threads. For fair comparison, calculate an average (~3) of the execution times for each test case. Either single- thread or multi-thread computation should be enabled in each test but not both.

[1] https://www.programiz.com/java-programming/examples/calculate-methods-execution-time

What to submit

- Your java programs
- A report explaining the implementation of the single-thread and multi-thread implementation, with reference to the relevant lines of your code. The report should include evidence of successful implementation. The execution times should be given numerically and displayed graphically using charts with associated critical discussion. Maximum 2-3 pages.
- You must include your repository (attempts starting from week 5, May 13th, weekly update)

Assessment Criteria

- Correctness and efficiency of the implementations code
- Discussion quality of the testing results