NGC (normalized grey-scale correlation) Image Recognition Library for Delphi

Overview

This library is the only native Delphi library for precise image recognition. The library can find a small image (“pattern”) within a larger image (“main image”).

NGC stands for “normalized grey-scale correlation” which is:

$$\rho(x,y) = \frac{\sum_{u=-N}^{N}\sum_{v=-M}^{M}\left(f(x'+u,y'+v)-\bar{f}\right)\cdot\left(g(x+u,\ y+v)-\ \bar{g}\right)}{\sqrt{\sum_{u=-N}^{N}\sum_{v=-M}^{M}\left(f(x'+u,y'+v)-\bar{f}\right)^{2}\cdot\sum_{u=-N}^{N}\sum_{v=-M}^{M}\left(g(x+u,\ y+v)-\ \bar{g}\right)^{2}}}$$

Usage examples

1. Natural images: You can search if a specific car is present in a image or not.
2. Generated images: Finding out if a check box is checked or not in an input image. The image can be a screenshot (in this case you get 100.0% accuracy) of a program or a scan of a paper form.
3. OCR – Finding text in an image. You can search a whole string (fast), or you can search each character in the alphabet (slow). This way you can basically build an OCR program.

etc

Is this AI image recognition?

No. This is different than AI image recognition algorithms. An AI will recognize all type of cars in the main image.

This NGC algorithm will recognize only one pattern (car). If you have 100 types of cars, you need to search each one individually (therefore, slower than the AI).

However, compared with the AI where you have to pay for each image you process, this NGC algorithm has no additional costs.

Input

Input source

The output quality of the algorithm is (naturally) directly proportional to the quality of the input image:
1. Screenshots: Provide optimal recognition conditions with 100% accuracy.

2. Images scanned with a scanner: Very very good accuracy when sourced from flatbed scanners.

3. Smart phone: The algorithm works also on noisy images (images that have been acquired with photo cameras instead of scanners) but its accuracy will be lower.

Storage type (recommendation)

1. Non-loosy compression: To prevent further degradation of your input, store your images under non-lossy compression algorithms (such as zipped bmp, bmp, png, tiff).
2. Lossy compression: Lossy compression algorithms (such as jpeg, gif) can dramatically affect the quality of the input image. Of course, once you converted the image to JPG, the damage was done – the quality was lost forever even if you convert your image back to bmp or png.

The recommended storage format is PNG.

Input format

The accepted input type of the algorithm is grayscale 8-bit bitmaps. Of course, you can write a small routine to convert your input from png, jpg, etc to bmp.

Input flexibility

The algorithm allows for some variations, so the pattern image would be found even if it were slightly rotated (maximum 20 degrees) or resized (maximum 30%).

Speed

For the input image below, the speed of the program (in debug mode) is 4.1 seconds (on a 12-years old AMD FX 8350). The time increases (of course) exponentially with the size of the image. If you have really large images, you can of course scale them down.

The speed of the algorithm can be dramatically improved by several techniques (downsizing the image, pyramid search, multi-threading, caching some of the calculations). I will try to implement these in the next version of the algorithm. This could improve the speed at least 30 times making the algorithm more than suitable for real-time image processing.

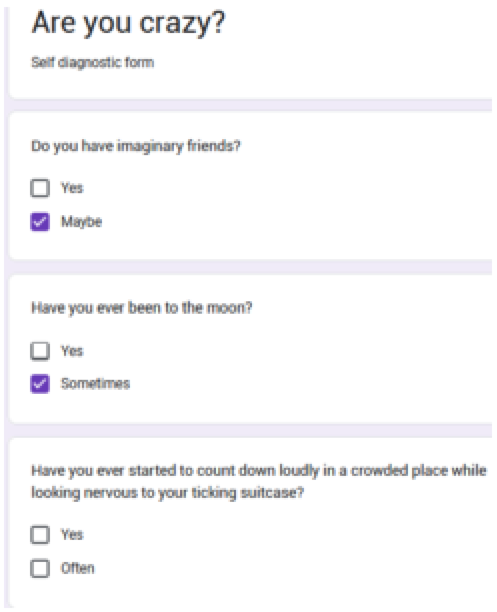
Note that there are other ways to dramatically speed up the algorithm (by cutting some corners) but they will result in less than 100% accuracy.

The **ideal** scenario is when you don't need to process images in real time. In this case, you can simply process the images in background and store the results (the coordinates where the pattern was detected in the main image) somewhere to a file (INI file, etc). This way, you can simulate real-time performance.

Demo program

As a concrete example, let's say we want to detect how many check boxes are checked in this "Are you crazy – Self diagnostic test".

We grab a screenshot of the form as completed by the patient:



Are you crazy?

Self diagnostic form

Do you have imaginary friends?

- ☐ Yes
- ☑ Maybe

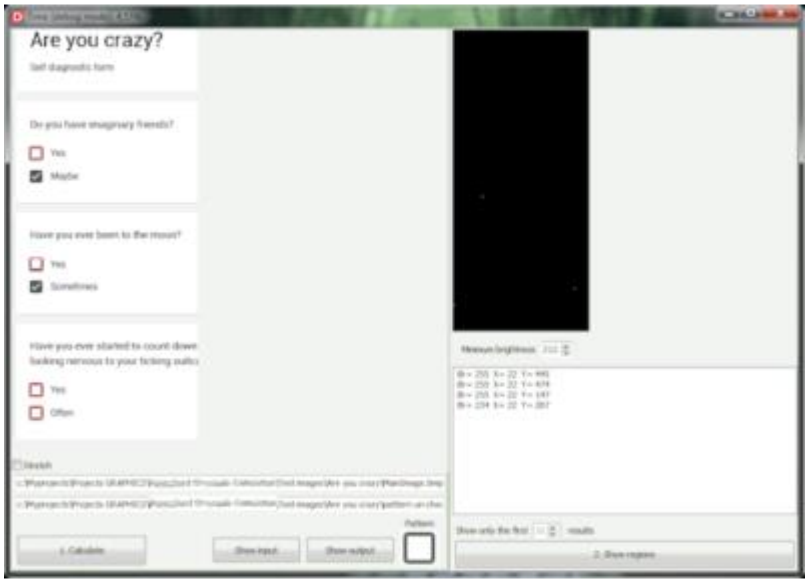
Have you ever been to the moon?

- ☐ Yes
- ☑ Sometimes

Have you ever started to count down loudly in a crowded place while looking nervous to your ticking suitcase?

- ☐ Yes
- ☐ Often

We load the above screenshot into the program:



Are you crazy?

Self diagnostic form

Do you have imaginary friends?

☐ Yes

☑ Maybe

Have you ever been to the moon?

☐ Yes

☑ Sometimes

Have you ever started to count down looking nervous to your ticking [illegible]

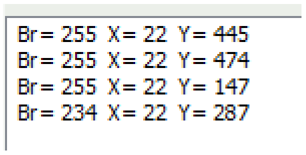
☐ Yes

☐ Often

Of course, we need to also load a small crop the check box. A checkbox can have two states (checked/unchecked). Therefore we will have two “pattern” images”: ☑☐.

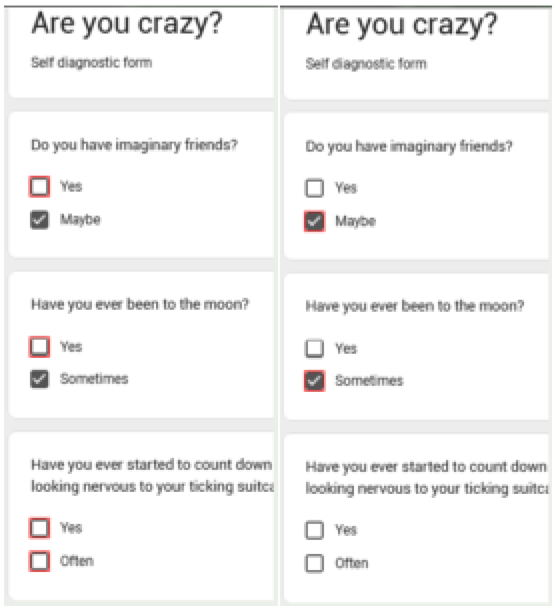
The algorithm has to be executed for each pattern (checkbox state).

After we run the program, it will indicate the coordinates where the check boxes have been detected:



```
Br= 255  X= 22  Y= 445
Br= 255  X= 22  Y= 474
Br= 255  X= 22  Y= 147
Br= 234  X= 22  Y= 287
```

And as visual feedback it will also show in red rectangles the position of the check boxes:


Are you crazy?
Self diagnostic form

Do you have imaginary friends?
Yes [✓ Maybe]

Have you ever been to the moon?
Yes [✓ Sometimes]

Have you ever started to count down looking nervous to your ticking suitc
Yes Often

Are you crazy?
Self diagnostic form

Do you have imaginary friends?
Yes [✓ Maybe]

Have you ever been to the moon?
Yes [✓ Sometimes]

Have you ever started to count down looking nervous to your ticking suitc
Yes Often

The program can process thousands of such forms per hour.

Techniques to improve the accuracy

If you have noisy or low-quality inputs, employ preprocessing methods to enhance accuracy:

- Straightening misaligned images.
- Deblocking and denoising.
- Removing compression artifacts.
- Adjusting contrast and brightness.

Delphi compatibility

Written and tested in Delphi 10.4 / Delphi 11, but it should work without any modifications on all Delphi editions that support inline variables. If you have an older edition you need to re-declare the variables (5 minutes work). The code should work also under Lazarus/FPC but I have never tested it.

The library has been tested with FastMM4 and has no memory leaks.

This NGC library needs some functions from my LightSaber library, in order/process to open images.

If you want, you can remove this dependency (you can write your own image opening routines – or you can extract them from LightSaber).