Haskell: Functional Programming
Final Project Report

Due on Jan 17, 2016

Chen Zijian
THSS32
2013013340

1. Introduction

This project is built for sub-image location, which is to find the position of a small sub-image appearing at its parent image. Input an image A and its sub-image B, the program will show you the position of B's top left-hand corner at which it is located in A. The sub-image may be a pollution image, e.g with noise or taint. My project can deal with all the 4 kinds of interference mentioned in the document (i.e: linear filtering, color enhancement, impulse noise, and taint).

2. Basic Idea

It's easy for human beings to read images, but it's hard for computers. However, it's hard for human beings to understand signals, but it's easy for computers. With this basic idea, we can regard the images as a tow-dimensional signal and applying standard signal-processing techniques to it. I'll introduce the algorithms in part 3.
The input image may be polluted, so is the signal. If we can reduce the noise of the signal, we can ignore the pollution of the image. In that case, we can use the same algorithm to deal with different kinds of pollution. That is exactly what I have done in this project.

3. Core Algorithms

3.1 Cross-correlation

In signal processing, cross-correlation is a measure of similarity of two series as a function of the lag of one relative to the other. It is commonly used for searching a long signal for a shorter, known feature.[1]
Regarding images as signals, we can use it to search a large picture for a smaller, known sub-image. *If we have two images A and B of the same size then the cross-correlation image C is defined as*

$$C(\delta_x, \delta_y) = \sum_{x=0}^{sx-1} \sum_{y=0}^{sy-1} A_{x,y} B_{x,y}$$

If we use a Fourier transform on both images, we get the following equation:[2]

$$C(\delta_x, \delta_y) = \mathcal{F}^{-1}(\ \mathcal{F}(A) \times \mathcal{F}(A)^*)$$

The position (δ_x, δ_y) where C is maximal determines the position to have the best overlap between two images. So we can extend the sub-image with 0s and use the large image and the extended sub-image to compute the cross-correlation image, and the find the best position. This seems good. But that's not our case.
I did an experiment with the *img1* images, all the output position are the same but wrong. The result seems terrible, but at least we know that the method is anti-interference.

3.2 Mean filtering

In part 3.1, we just apply the equation to compute the cross-correlation matrix without preprocessing the images. That's the main reason we get the wrong answer. So I just modify the equation a bit:

$$C(\delta_x, \delta_y) = \mathcal{F}^{-1}(\ \mathcal{F}(\mathcal{N}(A)) \times \mathcal{F}(\mathcal{N}(A))^*)$$

where $\mathcal{N}$ is an image preprocessing function.
Think that we have two signals, one is $sinx$, the other is $sinx+1$. They have the same feature but different averages. So we have to substract their average. For images, we do the same. First, we do a mean filtering to get mean filtering matrix, and then we substract the matrix from the original image. Finally we put the preprocessed images into the equation ot get the answer.

4. Difficulties

4.1 Haskell itself

Haskell itself is a difficulty. Since not many people use haskell, there aren't some libraries such as opencv, and some libraries are slow.

4.2 Image Preprocessing

I give an explation of why I use mean filtering to preprocess the images. But before I got the explanation, I didn't know why it didn't work until i read a paper. Besides, the mean filtering function is difficult to write. I really didn't want to write it, but the mean filtering function of the library I use is slow and I can't use library to deal with the core of the problem.

5. Highlights

5.1 Problem Shift

I shift the problem of image processing to signal processing. And I look at the problem from a different view from other students(I think).

5.2 Mean Filtering

I implemented a mean filtering function which is faster than the mean filtering function provided by the library Friday I use. The function

```
meanFilter' ::  (Int, Int) -> I.Grey -> I.Manifest (Double)
```

5. Highlights continued on next page...

is written by myself, while
`meanFilter :: (Int, Int) -> I.Grey -> I.Manifest (Double)`
is the one that is based on the library. My own version's complexity is just $\Theta(n)$(n is the image size). It is unrelevant to the window size.

5.3 One Method, All Interference

I have solved all of the 4 kinds of interference with only one method.
And although I have not tested yet, I think it can deal with some other kinds of interference.

5.4 Accurate and Fast

My solution is accurate. I tested all the given images, and get the right answers. As for speed, as mentioned above, the complexity of mean filtering is $\Theta(n)$. The complexity of Fast Fourier transform is $\Theta(nlogn)$. Thus the whole complexity of my solution is just $\Theta(nlogn)$(n is the parent image's size). I also do the two images' preprocessing in parallel, and use some compile argument and run argument to let the compiler help me to speed the program.

6. Usage

Compile the code with command: `ghc -threaded -rtsopts -O2 -XBangPatterns match.hs`
Run the program with command: `./match <parent-image name> <sub-image name> <mode number> +RTS -N2 -H -K512M -ki256M -A512M -I0`

7. Package

1. carray-0.1.6.3
2. fft-0.1.8.2 (This package also required a C library FFTW to be installed)
3. friday-0.2.2.0
4. friday-devil-0.1.1.1 (This package also required a C library DevIL to be installed)

8. References

[1] Wikipedia "Cross-correlation"
[2] W. Van Belle "An adaptive filter for the correct localization of subimages: FFT based subimage localization requires image normalization to work properly", Signal Process., pp.11 2007