Interactive Functional Medical Image Analysis

A demonstration using Racket and the *vigracket* library to detect sickle-cell anaemia

Benjamin Seppke
University of Hamburg
Dept. Informatics
Vogt-Kölln-Str. 30
Hamburg, Germany 22527
seppke@informatik.uni-hamburg.de

Leonie Dreschler-Fischer
University of Hamburg
Dept. Informatics
Vogt-Kölln-Str. 30
Hamburg, Germany 22527
dreschler@informatik.uni-hamburg.de

ABSTRACT

This article demonstrates the functional application of Computer Vision methods by means of a prototypical process chain. For this demo, we have selected the application area of medical image analysis, in detail the classification of blood cells using microscopic images. We further focus on the task of detecting abnormal sickle-shaped red blood cells, which are an indicator for a relatively common disease in countries around the equator: the so-called sickle-cell anaemia. From the functional languages, we have chosen Racket and the *vigracket* Computer Vision library [3]. Although this demo just scratches the surface of medical image processing, it provides a good motivation and starting point.

CCS CONCEPTS

•Computing methodologies →Image segmentation; •Software and its engineering →Functional languages; •Applied computing →*Bioinformatics;*

KEYWORDS

Functional Programming, Racket, Medical Image Processing, Computer Vision, Language Interoperability

ACM Reference format:
Benjamin Seppke and Leonie Dreschler-Fischer. 2017. Interactive Functional Medical Image Analysis. In *Proceedings of European Lisp Symposium 2017, Brussels, Belgium, April 2017 (ELS'2017)*, 2 pages.

1 INTRODUCTION

Functional programming and functional languages have a long tradition in research and teaching at the Department of informatics at the University of Hamburg. Since 8 years, we successfully combine the Computer Vision library *vigra* [1] with functional programming languages. As an example in [4] we have presented how well Computer Vision approaches may be introduced into the Racket programming language. To reach a broader range of users, it shall not be unmentioned that the used C-wrapper library in platform-independent and has been tested and proven to work under Windows, Mac OS X and Linux.

After the latest optimisations, the Computer Vision wrapper libraries are even more powerful with respect to performance and image segmentation tasks [2]. Functional languages natively offer interactive development cycles, generic modelling and probably the most powerful garbage collectors.

ELS'2017, Brussels, Belgium 2017.

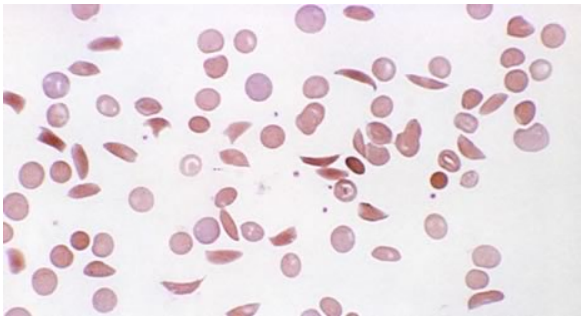

Figure 1: Blood sample of a patient with sickle-cell anaemia. The sickle-shaped cells occur among the regular, circle-shaped cells. (©Getty Images/Photoresearchers)

2 DEMONSTRATION

Sickle-cell anaemia or sickle-cell disease is genetic disease occurring in many parts of Africa and other countries. Besides its negative effects, there also seems to be a protective effect against Malaria. A comprehensive report on the sickle-cell disease and on the research progress can be found in [5]. We have chosen the detection of the sickle-cell disease for this demonstration as it requires shape detection and description of red blood cells on microscopy images. As an example, Figure 1 shows a typical image of a patient's red blood cells. For this demonstration we will distinguish between the main working phases of the process chain in logical order:

2.1 Loading images

Before loading images, it might be useful to define a working folder. By changing this folder, we can easily adapt our demo to different users or other folders. The image is then be loaded by *vigracket* in a three element list containing arrays of foreign memory for the red, green and blue channel.

```
(require vigracket)

(define dir (current-directory))
(define img (loadimage (build-path dir "cells.jpg")))
```

2.2 Preprocessing

The first step for the analysis of the different imaged blood cells is the division of the image contents into foreground (the cells) and background. This can e.g. be achieved by applying a threshold to one or more channels of the image.

Table 1: Semantics of the columns for RGB region-wise feature extraction. Each row corresponds to one segment.

Columns	Features
0	segment size
1, 2	upper left x and y-coordinates of segment
3, 4	lower right x and y-coordinates of segment
5, 6	mean x and y-coordinates of segment
...	other statistics of segment

The threshold is then used to pre-classify the pixels of the image into foreground (grey value = 255) and background pixels (grey value = 0). Here, we have chosen the red-channel and a fixed threshold of 222 to generate the mask, described above. After the thresholding we apply a morphologic opening filter to suppress the influence of smaller artefacts:

```
(define mask (image-map (λ(x) (if (< x 222.0) 255.0 0.0))
                        (image->red img)))
(define omask (openingimage mask 1.0))
```

2.3 Divison in segments

To analyse the single cells' properties, we have to divide the foreground into corresponding segments. If we define a segment as a connected component of (masked) pixels, we can assign unique labels for each segment. The `#t` tells the function to use eight-pixel connectivity for the component detection and `0.0` denotes the background value:

```
(define labels  (labelimage omask #t 0.0))
```

Instead of 0 and 255, this function assigns increasing values from 1 to the number of connected components found for each foreground classified pixel.

2.4 Segment analysis and classification

To decide whether a segment represents a sickle-cell or an circle-shaped cell, we need to get statistics, also called features, for each segment. This process can be automised by vigracket's latest functional extensions. We use the `extractfeatures` method to derive RGB-based statistics using the image and the label image.

```
(define stats (extractfeatures img labels))
```

The result, `stats`, is also an image, but with slightly changed semantics. Each row represents on region with its extracted features. The semantic with respect to the columns is shown in Table 1. For this demonstration, we find it sufficient to check whether the dimensions of each segment are roughly circle-like. We further use the aspect ratio of each region's bounding box plus one threshold, to accept a circle-like structure.

```
(define (circle-like? segment threshold)
    (let* [(width  (- (image-ref stats 3 segment 0)
                      (image-ref stats 1 segment 0)))
           (height (- (image-ref stats 4 segment 0)
                      (image-ref stats 2 segment 0)))
           (a (max width height))
           (b (min width height))]
      (< (/ a b) threshold)))
```

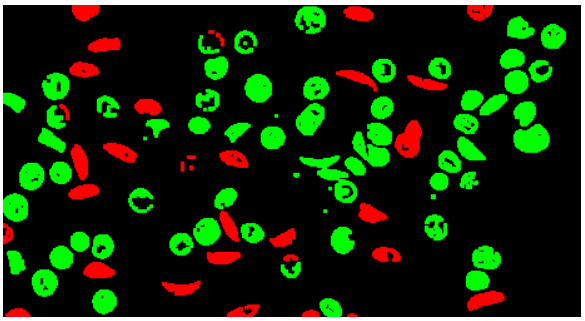

Figure 2: Classification result of Figure 1 using the demonstrated approach. Red: classified sickle-cells, green: classified circle-like cells, black: background.

2.5 Extraction and usage of results

Using the `circle-like?` function, we are now able to classify each region. Since the classification is a binary decision, we may simply filter the list of all segments:

```
(define maxlabel (image-reduce max 0.0 labels))
(define label-ids (build-list maxlabel values))
(define filtered-ids (filter (curryr circle-like? 4/3)
                             label-ids))
```

Now `filtered-ids` contains the ids of all segments which are circle-like (aspect ratio below 4/3) and thus not correspond to sickle-cells. This list can be used to quantify the ratio of different cell-types, to mark them in the image (see Figure 2) or to derive special statistics like color for these cells. Although the resulting classification leaves still place for improvements.

3 CONCLUSIONS

We demonstrated the use of interactive functional Computer Vision in a medical context. Although this demonstration has been performed using Racket and the *vigracket* library, it might also be performed by means of Common Lisp and the *vigracl* library.

Due to the limitations of this demo, the quality of the results is limited, too. However, this should not be seen as a comprehensive and best-possible segmentation and classification approach for the selected application area. Instead, it should be a motivation for all readers to utilise the power and simplicity of functional programming languages and powerful libraries for interdisciplinary areas of research. Although imperative languages are very popular at these fields at the moment, many tasks can be solved using functional languages while benefitting from their advantages, too.

REFERENCES

[1] Ullrich Köthe. 2017. The VIGRA homepage. Retrieved January 30, 2017. (2017). http://ukoethe.github.io/vigra/

[2] Benjamin Seppke. 2016. *Near-Realtime Computer Vision with Racket and the Kinect sensor.* Technical Report. University of Hamburg, Dept. Informatics.

[3] Benjamin Seppke. 2017. The vigracket homepage. Retrieved January 30, 2017. (2017). https://github.com/bseppke/vigracket

[4] Benjamin Seppke and Leonie Dreschler-Fischer. 2015. Efficient Applicative Programming Environments for Computer Vision Applications: Integration and Use of the VIGRA Library in Racket. In *Proceedings of the 8th European Lisp Symposium.*

[5] Graham R. Serjeant. 2010. One hundred years of sickle cell disease. *British Journal of Haematology* 151, 5 (2010), 425–429. DOI: http://dx.doi.org/10.1111/j.1365-2141.2010.08419.x