Generische Bildverarbeitung mit VIGRAnumpy

Benjamin Seppke
14.11.2013

Inhalt

- Einführung
- Konzepte der VIGRA numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Inhalt

- Einführung
- Konzepte der VIGRA numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Einführung

- Noch eine Bildverarbeitungsbibliothek
- Bisherige Bibliotheken vor allem „fertige Werkzeuge“
- 

: Generische Basis-Bausteine, aus denen vorhandene oder neue Werkzeuge erstellt werden können!

VIGRA stands for "Vision with Generic Algorithms". It's a novel computer vision library that puts its main emphasis on customizable algorithms and data structures. By using template techniques similar to those in the C++ Standard Template Library, you can easily adapt any VIGRA component to the needs of your application, without thereby giving up execution speed.

http://hci.iwr.uni-heidelberg.de/vigra

VIGRA-Bibliothek

- C++-Bibliothek (aktuell: Version 1.9.0)
  - effiziente Implementierung
  - effizienter Algorithmen
- (Recht) Exzessive C++-Template-Nutzung
- Für unser Projekt: Zeit begrenzt, somit ist kein umfassender Einstieg in C++ möglich ☹
- Daher:

  VIGRA numpy für den Zugriff der VIGRA-Funktionen aus Python/NumPy heraus! ☺

Weitere Informationen zu VIGRA numpy

- Benötigte Software(-pakete)
  - Python
  - Numpy
  - Boost::Python
  - FFTW3/FFTW3f
  - Optional:
    - Bildformate: Tiff, Jpeg, PNG, OpenEXR, HDF5, h5py
    - Dokumentationsgeneratoren: sphinx, doxygen
- Dokumentation:
  http://hci.iwr.uni-heidelberg.de/vigra/doc/vigranumpy/index.html

Inhalt

- Einführung
- Konzepte der 

numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Repräsentation von Bilddaten

- Wie werden Bilder repräsentiert?
  → Arrays!
- Aber: Reihenfolge ist of nicht einheitlich geregelt!
- Beispiele:
  - Img(x,y) → Img[x][y] oder Img[y][x]?
  - Img(x,y,t) → Img[t][x][y] oder Img[t][y][x] oder Img[x][y][t] oder Img[y][x][t]?
- Unterschied nicht nur für Semantik, sondern auch für Geschwindigkeit wichtig!
- Numpy-Arrays führen keine semantischen Informationen mit sich!

Konzept der axistags

- VigraArray erbt von NumpyArray!
- Enthält zusätzlich sog. „axistags“
  - Vordefinierte Tags: x, y, z, c, t

    Entspr: (x,y,z)-Achsen, c=Farbachse, t=Zeitachse
- Numpy-Arrays können einfach konvertiert werden:
  - Img = Image(numpyImgArray),

    wobei numpyImgArray 2D oder 3D sein kann.

Vorteile der axistags

- Bilddaten bleiben prinzipiell in NumPy-Arrays gespeichert
- Semantische Ungereimtheiten werden ausgemerzt
- Transparente Ausführung der Algorithmen.
  Beispiel:
  - NumPy-Array mit [x, y]-Achsendefinition (Fortran-Order)
  - Unter Angabe der korrekten axistags:
    1. Konvertierung in C-order
    2. Ausführung in C++
    3. Zurück-Konvertierung in Fortran-Order

Inhalt

- Einführung
- Konzepte der VIGRA numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Import/Export (vigra.impex)

- Unterschiedliche Formate verfügbar, je nach Build-Abhängigkeiten:
  - readImage
  - writeImage
  - readVolume
  - writeVolume
- Zusätzlich ggf. noch das Datenformat HDF5:
  - readHDF5
  - writeHDF5

Array-Arithmetik

- Reimplementiert, um die axistags zu unterstützen:

absolute absolute add arccos arccosh arcsin arcsinh arctan arctan2 arctanh bitwise_and bitwise_or bitwise_xor ceil conjugate conjugate copysign cos cosh deg2rad degrees divide equal exp exp2 expm1 fabs floor floor_divide fmax fmin fmod frexp greater greater_equal hypot invert invert isfinite isinf isnan ldexp left_shift less less_equal log log10 log1p log2 logaddexp logaddexp2 logical_and logical_not logical_or logical_xor maximum minimum modf multiply negative nextafter not_equal ones_like power rad2deg radians reciprocal remainder remainder right_shift rint sign signbit sin sinh spacing sqrt square subtract tan tanh true_divide trunc

- Teilweise auch direkt am Array-Objekt aufrufbar!

__abs__ __add__ __and__ __div__ __divmod__ __eq__ __floordiv__ __ge__ __gt__ __invert__ __le__ __lshift__ __lt__ __mod__ __mul__ __ne__ __neg__ __or__ __pos__ __pow__ __radd__ __radd__ __rand__ __rdiv__ __rdivmod__ __rfloordiv__ __rlshift__ __rmod__ __rmul__ __ror__ __rpow__ __rrshift__ __rshift__ __rsub__ __rtruediv__ __rxor__ __sub__ __truediv__ __xor__

Filter (vigra.filters)

- Faltung mit variablem Kernel
  - Allgemeine 2d-Faltung
  - Separierte 2d-Faltung
- Vordefinierte Faltungsfunktionen (Beispiele)
  - Img_dest = gaussianSmoothing(Img_src, **sigma**)
  - Img_dest = gaussianGradientMagnitude(Img_src, **sigma**)
  - Img_dest = gaussianSharpening2D(Img_src,factor)
- Morphologische Operationen
  - Erosion/Dilatation
  - Opening/Closing
- (Struktur-)Tensoren uvm.

Bildraum-Transformationen (vigra.sampling)

- Bilder vergrößern/verkleinern:
  - Img_dest = resampleImage(Img_src, factor)
  - Img_dest = resamplingGaussian(Img_src, …)
  - Img_dest = resize(Img_src, shape, interpolationOrder)
- Bilder rotieren
- SplineImageView für Subpixel-Bildzugriff.
  Beispiel:
  - SplView = SplineImageView[0,1,2,3,4,5](image)
  - SplView(12.23231, 23.23231) = 231.1242398
- Bildpyramiden uvm.

Bildanalyse (vigra.analysis)

- Canny-Kantendetektor
- Feature-Extraktion
- Zusammenhangskomponenten finden
  - Bilder (4er und 8er Nachbarschaften)
  - Volumen (6er und 26er Nachbarschaften)
  - Regionen statistisch untersuchen
  - Regionen verschmelzen
  - Crack-Edge-Repräsentation
- Wasserscheiden-Transformation

Weitere Module

Heute leider nicht besprechen können wir:

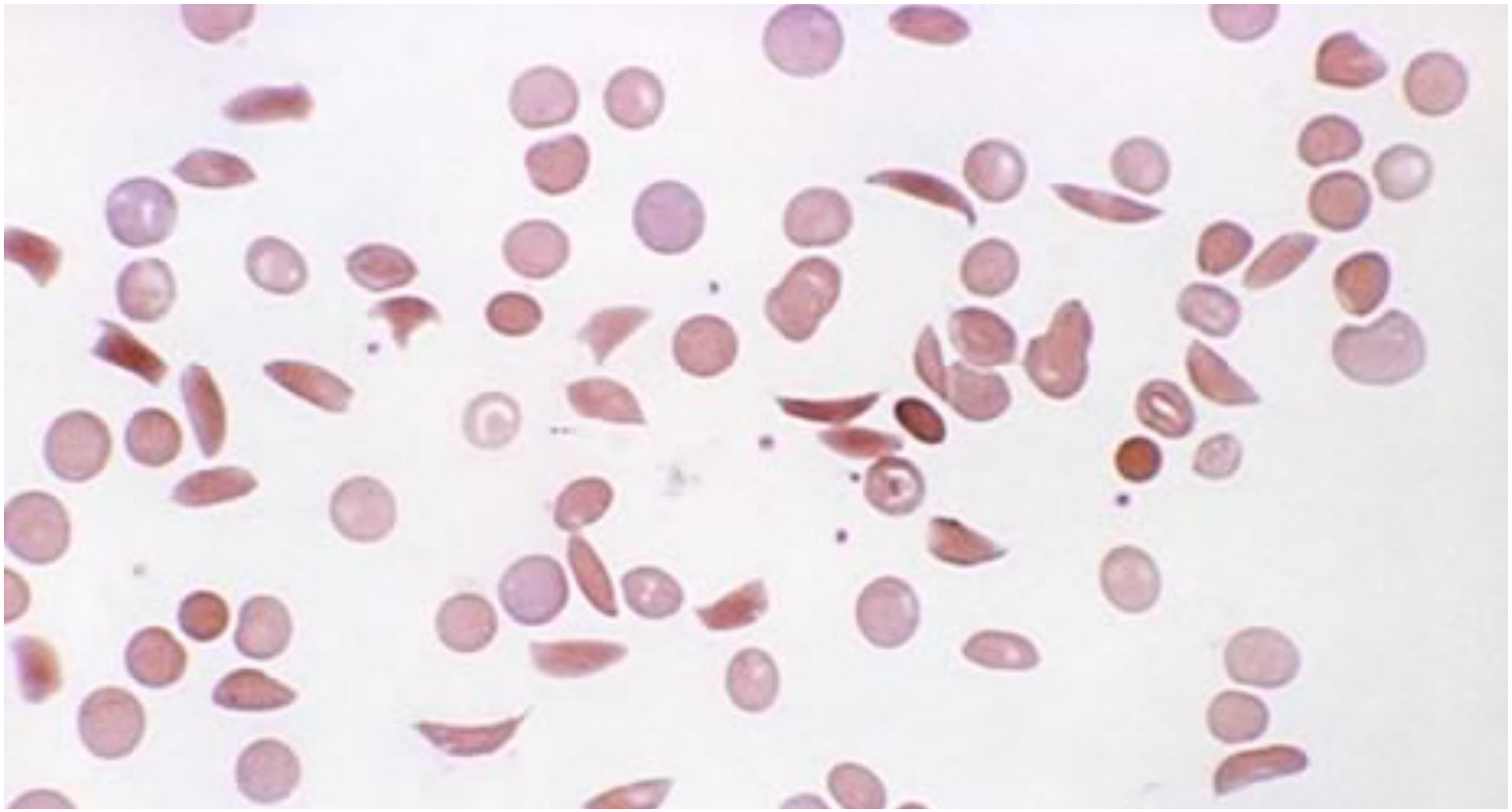
- Farbberechnungen und Farbraummanipulationen (vigra.colors)
- Fourier-Transformation (vigra.fourier)
- Geometrische Operationen (vigra.geometry)
- Optimierungsverfahren (vigra.optimization)
- Maschinelles Lernen (vigra.learning)
- Rauschschätzung und Normalisierung (vigra.noise)

Inhalt

- Einführung
- Konzepte der 

numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Beispiel aus der Medizin: Sichelzellenanämie



1. Laden und Vorverarbeitung

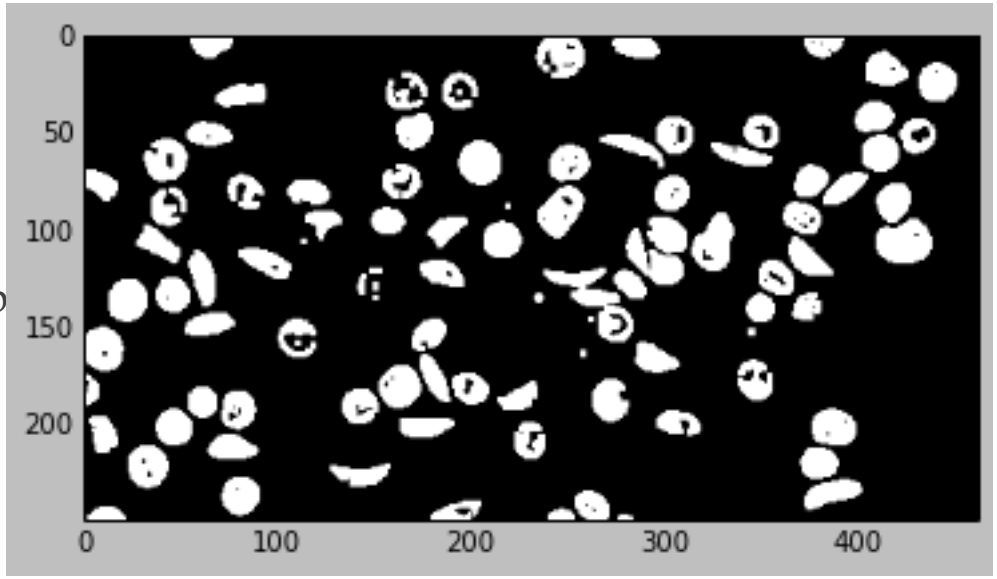
```
import vigra as vi
import numpy as np

img = vi.readImage('sichelzellen.jp
vi.imshow(img)

i_img = vi.Image(img[...,0])
vi.imshow(vi_img)

vi_img_t = vi_img < 222
vi.imshow(vi_img_t)

vi_img_to = vi.filters.discOpening(vi_img_t.astype(np.uint8), 1)
vi.imshow(vi_img_to)
```



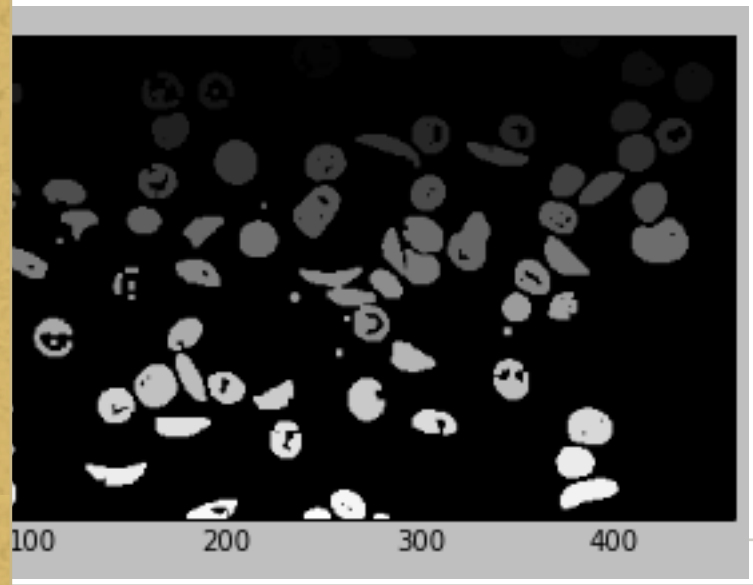
2. Finden von Zusammenhangskomponenten und Extraktion von Regionsmerkmalen

```
vi_label_img = vi.analysis.labelImageWithBackground(vi_img_to)
vi.imshow(vi_label_img)

vi.analysis.supportedRegionFeatures(vi_img, vi_label_img)

features = vi.analysis.extractRegionFeatures(vi_img, vi_label_img)
region_count = features.maxRegionLabel()+1
```

['Central<PowerSum<2/3/4> >', 'Coord<ArgMaxWeight >',
'Coord<ArgMinWeight >',
'Coord<DivideByCount<Principal<PowerSum<2> > > >',
'Coord<Maximum >', 'Coord<Minimum >', 'Coord<PowerSum<1> >',
'Coord<Principal<Kurtosis > >', 'Coord<Principal<PowerSum<2> > >',
'Coord<Principal<PowerSum<3> > >', 'Coord<Principal<PowerSum<4> >
>', 'Coord<Principal<Skewness > >', 'Count', 'Global<Maximum >',
'Global<Minimum >', 'Histogram', 'Kurtosis', 'Maximum', 'Mean',
'Minimum', 'Quantiles', 'RegionAxes', 'RegionCenter', 'RegionRadii',
'Skewness', 'Sum', 'Variance',
'Weighted<Coord<DivideByCount<Principal<PowerSum<2> > > > >',
'Weighted<Coord<PowerSum<1> > >',
'Weighted<Coord<Principal<Kurtosis > > >',
'Weighted<Coord<Principal<PowerSum<2/3/4> > > >',
'Weighted<Coord<Principal<Skewness > > >', 'Weighted<PowerSum<0>
>', 'Weighted<RegionAxes>', 'Weighted<RegionCenter>',
'Weighted<RegionRadii>']



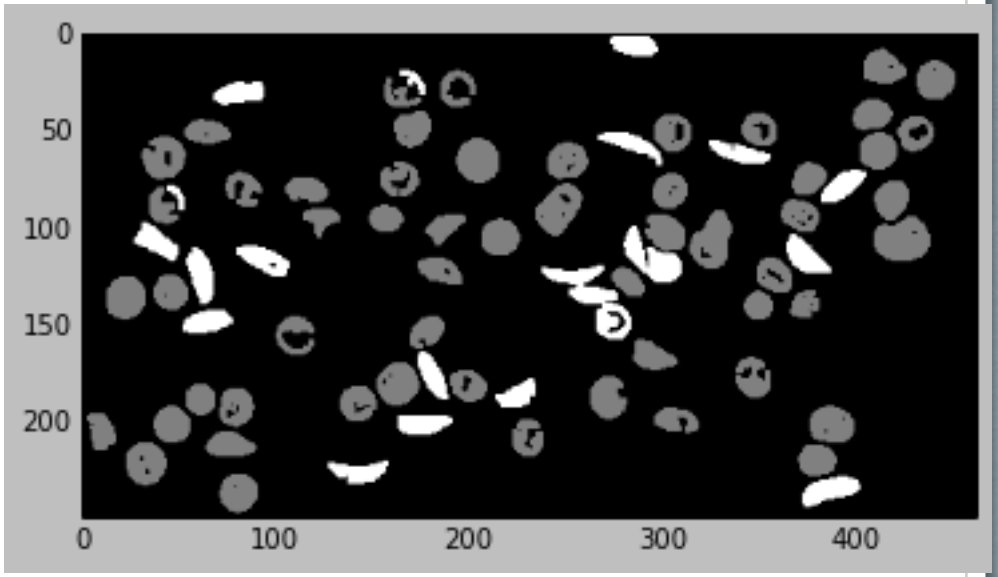
3. Klassifikation

```
...
bboxes = zip(features['Coord<Minimum >'],
             features['Coord<Maximum >'])

final_features=np.zeros(region_count)
vi_img_res = vi_img_to.copy()

for i in range(region_count):
    if regionAtBorder(img, bboxes[i]):
        final_features[i] = 0
    elif features['Count'][i] < 50:
        final_features[i] = 0
    elif regionNotCircleLike(features['RegionRadii'][i]):
        final_features[i] = 2
    else:
        final_features[i] = 1
    w = np.where(vi_label_img ==i)
    vi_img_res[w] = final_features[i]

vi.imshow(vi_img_res)
```




```
def regionAtBorder(img, region_bbox):
    ul, lr = region_bbox[0], region_bbox[1]
    return      (ul[0]==0 or ul[1]==0)
                or (lr[0] == img.width-1 or lr[1] == img.height-1)
```

```
def regionNotCircleLike(region_radii, epsilon=2):
    return region_radii[0]/region_radii[1] > epsilon
```

Inhalt

- Einführung
- Konzepte der VIGRA numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

VIGRAnumpy erweitern

- Eigene Erweiterungen: C/C++ mit Boost::Python
- Vorgefertigte Wrapper-Funktionen verfügbar
  - NumpyAnyArray
  - NumpyArray<3, Type>
  - NumpyArray<3, Multiband<Type> >
  - NumpyArray<2, Singleband<Type> >
- Ebenfalls vorhanden: reshape-Funktionen anhand der axistags
- Volle Template-Unterstützung auf C++-Seite
- Etwas komplexerer Code, Basis-Beispiel aber vorhanden

Inhalt

- Einführung
- Konzepte der VIGRA numpy
- Einblick in den Funktionsumfang
- Beispielhafte Bildverarbeitungsaufgabe
- Eigene Erweiterungen
- Zusammenfassung

Zusammenfassung

- 

als generische Bildverarbeitungsbibliothek
- 

numpy als unser Zugriffspunkt
- Viele generische, erprobte und theoretisch fundierte Werkzeuge
- Gutes „TestBed“ für neue Algorithmen
- Praktisch einsetzbar
- Mit Boost::Python ohne großen Aufwand erweiterbar

Vielen Dank für die Aufmerksamkeit!

- Zeit für Fragen, Anregungen und weitere Diskussionen