Kurzeinführung in die Bildverarbeitungsbibliothek

„LTILib"

Prof. Dr.-Ing. Andreas Meisel

1. Einführung

Visual-C-Projektstruktur ist vorgegeben

- Gearbeitet wird ausnahmsweise auf dem lokalen Rechner (wg. Geschwindigkeit).
- Das lokale Verzeichnis (s.u.) ist freigegeben.

Arbeitsschritte:

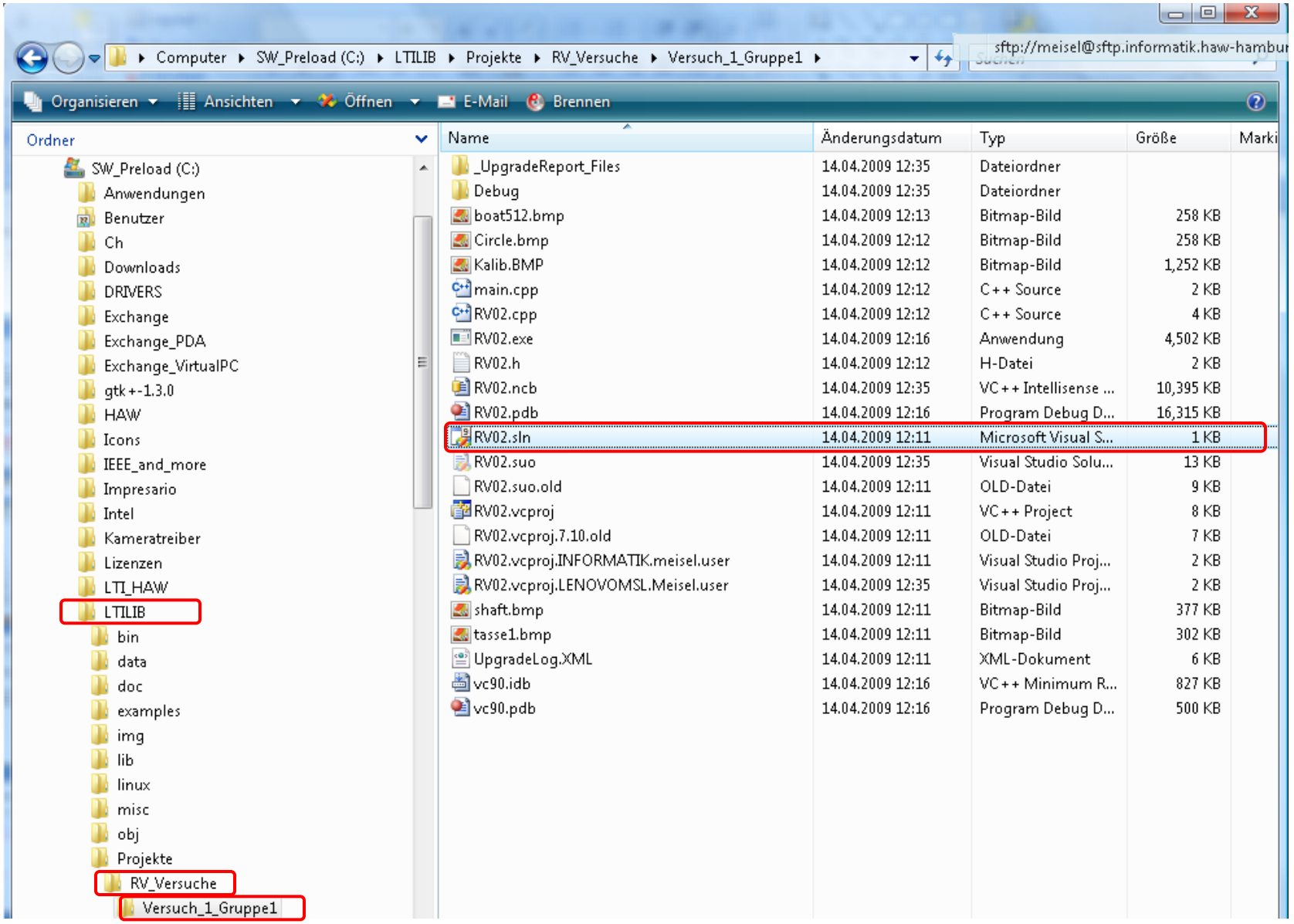
Projektverzeichnis aus Pub-Verzeichnis laden

```
/meisel/pub/Vorl_RobotVision/RobotVision_Praktikum/
Versuch_1_Gruppe1
```

nach

```
C:\LTILib\Projekte\RV_Versuche\Versuch_1_Gruppe1
```

- Angegebene Struktur unbedingt einhalten, sonst müssen alle Projektpfade geändert werden → viel unnötige Arbeit.
- zum Praktikumsende Projektverzeichnis in den privaten Sicherungsbereich retten


Computer ▸ SW_Preload (C:) ▸ LTILIB ▸ Projekte ▸ RV_Versuche ▸ Versuch_1_Gruppe1 ▸
sftp://meisel@sftp.informatik.haw-hambur
Organisieren
Ansichten
Öffnen
E-Mail
Brennen
Ordner
SW_Preload (C:)
Anwendungen
Benutzer
Ch
Downloads
DRIVERS
Exchange
Exchange_PDA
Exchange_VirtualPC
gtk+-1.3.0
HAW
Icons
IEEE_and_more
Impresario
Intel
Kameratreiber
Lizenzen
LTI_HAW
LTILIB
bin
data
doc
examples
img
lib
linux
misc
obj
Projekte
RV_Versuche
Versuch_1_Gruppe1
Name
Änderungsdatum
Typ
Größe
_UpgradeReport_Files 14.04.2009 12:35 Dateiordner
Debug 14.04.2009 12:35 Dateiordner
boat512.bmp 14.04.2009 12:13 Bitmap-Bild 258 KB
Circle.bmp 14.04.2009 12:12 Bitmap-Bild 258 KB
Kalib.BMP 14.04.2009 12:12 Bitmap-Bild 1,252 KB
main.cpp 14.04.2009 12:12 C++ Source 2 KB
RV02.cpp 14.04.2009 12:12 C++ Source 4 KB
RV02.exe 14.04.2009 12:16 Anwendung 4,502 KB
RV02.h 14.04.2009 12:12 H-Datei 2 KB
RV02.ncb 14.04.2009 12:35 VC++ Intellisense ... 10,395 KB
RV02.pdb 14.04.2009 12:16 Program Debug D... 16,315 KB
RV02.sln 14.04.2009 12:11 Microsoft Visual S... 1 KB
RV02.suo 14.04.2009 12:35 Visual Studio Solu... 13 KB
RV02.suo.old 14.04.2009 12:11 OLD-Datei 9 KB
RV02.vcproj 14.04.2009 12:11 VC++ Project 8 KB
RV02.vcproj.7.10.old 14.04.2009 12:11 OLD-Datei 7 KB
RV02.vcproj.INFORMATIK.meisel.user 14.04.2009 12:11 Visual Studio Proj... 2 KB
RV02.vcproj.LENOVOMSL.Meisel.user 14.04.2009 12:35 Visual Studio Proj... 2 KB
shaft.bmp 14.04.2009 12:11 Bitmap-Bild 377 KB
tasse1.bmp 14.04.2009 12:11 Bitmap-Bild 302 KB
UpgradeLog.XML 14.04.2009 12:11 XML-Dokument 6 KB
vc90.idb 14.04.2009 12:16 VC++ Minimum R... 827 KB
vc90.pdb 14.04.2009 12:16 Program Debug D... 500 KB

Wichtig und nützlich: → zeigen

1. Compilieren des Projektes

2. Laufen lassen

3. - Bild vergrößern und verkleinern/
   - Grauwert ansehen
   - Kontrast erhöhen (scale MinMax)

Lösen lin. Gleichungssysteme mit MAPLE

T:\Maple.95\bin.win\cwmaple9.5.exe

```
> with(linalg):
Warning, the protected names norm and trace have been redefined and unprotected
> with(LinearAlgebra):
Warning, the name GramSchmidt has been rebound
> A1:=<<1,1,1>|<5,5,605>|<500,100,100>>;
```

$$A1 := \begin{bmatrix} 1 & 5 & 500 \\ 1 & 5 & 100 \\ 1 & 605 & 100 \end{bmatrix}$$

```
> b1:=<98,150,415>;
```

$$b1 := \begin{bmatrix} 98 \\ 150 \\ 415 \end{bmatrix}$$

```
> X1:=LeastSquares(A1,b1);
```

$$X1 := \begin{bmatrix} \frac{3859}{24} \\ \frac{53}{120} \\ \frac{-13}{100} \end{bmatrix}$$

2. Verwendete C++ - Mechanismen

2.1 Referenzen

Funktionsdefinition mit Zeigern

```
void swap(int *a, int *b){
    int Temp;

    Temp = *a;
    *a   = *b;
    *b   = Temp;
}
```

Funktionsdefinition mit Referenzen

```
void swap(int& a, int& b){
    int Temp;

    Temp = a;
    a    = b;
    b    = Temp;
}
```

Funktionsaufruf mit Zeigern

```
int x=10;
int y=20;


swap(&x, &y);
```

Funktionsaufruf mit Referenzen

```
int x=10;
int y=20;


swap(x, y);
```

2.2 Funktions-Templates

2.2.1 Motivation

Oft werden Funktionen benötigt, welche eine Funktionalität für verschiedene Datentypen anbieten.

Ansatz 1: Überladene Funktionen

```
int square(int a){
    return a*a;
}

double square(double a){
     return a*a;
}
```

Nachteile:

- Für jeden Datentyp muss eine Funktion definiert werden, obwohl der Code beider Funktionen praktisch gleich ist.
- Schlecht wartbar.

Ansatz 2: Makros

```
#define square(a) ((a)*(a))


k = square(12);
x = square(12.045);
```

Nachteile:

- Große Makros sind schwer debugbar.
- Keine Typüberprüfung.

2.2.2 Ansatz 3: Templates (= generische Funktionen)

Aus folgendem Code …..

```
template<class SimpTyp>
SimpTyp square (SimpTyp x){
     return (x*x);
}


int a = 10;
int b = square(a);
```

… erzeugt der Compiler:

```
int sqare(int a){
    return a*a;
}
```

Vorteile:

- Sourcecode nur einmal vorhanden.
- Typüberprüfung.

Verwendbarkeit:

- Dann dem „function overloading" vorzuziehen, wenn sich die Funktionen nur im Datentyp unterscheiden aber nicht im Code.

Ein **Funktions-Template wird definiert**, indem der Definition vorangestellt wird:

- das Schlüsselwort `template`
- anschließend folgen die in <> eingeschlossenen Parameter,
- die mit dem Schlüsselwort `class` oder `typename` beginnen.

Für die Typzuordnung sind nur die Funktionsparameter entscheidend, nicht der Rückgabewert.

2.3 Klassen-Templates

2.3.1 Motivation

Oft werden Klassen benötigt, welche eine Funktionalität für verschiedene Datentypen anbieten.

Beispiele:

- Stack für `char, int, float, double,` .....
- Queue für `char, int, float,` .....

Ein **Klassen-Template wird deklariert**, indem der Deklaration vorangestellt wird:

- das Schlüsselwort template
- anschließend folgen die in <> eingeschlossenen Parameter,
- die mit dem Schlüsselwort `class` oder `typename` beginnen.

Eine **Klassenmethode wird definiert**, indem der Definition vorangestellt wird:

- das Schlüsselwort template
- anschließend folgen die in <> eingeschlossenen Parameter,
- die mit dem Schlüsselwort `class` oder `typename` beginnen.

2.3.2 Instanziierung und Aufruf

Bei der Instanziierung von Objekten einer Template-Klasse wird die gewünschte Typausprägung des Objektes in <> geschrieben und dem Klassennamen angehängt.

```
// Instanziierung
Stack<int>  myStack;
int         a=10;


// Aufruf
myStack.push(a);
```

Vorteile:

- Sourcecode nur einmal vorhanden.
- Typüberprüfung.

2.4 Namensräume und der Scope-Operator

Namensräume sind Gültigkeitsbereiche für Bezeichner (z.B. Variablen-, Funktions-, und Klassennamen).

Typ. Anwendung: Ermöglicht die Mehrfachverwendung von Funktions- und Klassennamen.

```
namespace Counter32bit
{                                       // Codebereich wird unter den
   int   Count;                         // Namespace „Counter32bit"
   void  Reset() {Count=0;}             // gestellt.
   …
}

namespace Counter8Bit                   // Codebereich wird unter den
{                                       // Namespace „Counter8bit"
   char  Count;                         // gestellt.
   void  Reset() {Count=0;}
   …
}

Counter8Bit::Reset();  // Erst durch den Scope-Operator „::"
                       // wird klar welches Reset gemeint ist.
```

2.5 Iteratoren

Iteratoren sind Objekte, mit denen die Elemente eines Containers (z.B. list, queue, vector, …) sequenziell durchlaufen werden können.

Typ. Anwendung: Sequenzielle Abarbeitung aller Containerelemente

```
// Instanziierung
list<int>             myList
list<int>::iterator   it;
……

// Ausdrucken aller Listenelemente
for(it=myList.begin();  it != myList.end();  ++it)
        printf(“%d“, *it);
```

Der Zugriff auf die Listenelemente erfolgt mit **`*it`**.

3. Grundlegende Datentypen der LTILib

3.1 Geometrische Grundelemente (nur einige)

3.1.1 tpoint

- zweidimensionale Struktur, bestehend aus den Koordinaten x und y,
- Templateklasse, d.h. die Genauigkeit kann durch den Anwender festgelegt werden (int, float, double) ,
- Varianten sind `point` gleichbedeutend mit `tpoint < int >`
  `dpoint` gleichbedeutend mit `tpoint < double >`

Beispiel:

```
// Anlegen der Punkt-Objekte
tpoint<float>  ObjLocation;
point          UL, LR;

int            XCoord, YCoord;
...
// Verwenden der Punkt-Objekte
XCoord = UL.x;
YCoord = UL.y;
```

3.1.2 rectLocation

- Beschreibt rechteckige Regionen in Bildern (image, channel) durch:
  - Mittelpunkt-Position,
  - Orientierung,
  - die beiden Seitenlängen des Rechtecks.
- Besitzt verschiedene Methoden, z.B.
  `scale(..), shift(..), rotate(..), Area(..).`

Beispiel:

```
// Anlegen des rectLocation-Objektes
rectLocation   myBox;

// sonstige Deklarationen
tpoint<float>  Loc;
float          Ang, MaxL, MinL;
…
// Abfragen der Parameter des rectLocation-Objektes
Loc    = myBox.position;
Ang    = myBox.angle;
MaxL   = myBox.maxLength;
MinL   = myBox.minLength;
```

3.2 Konturen und Regionen (nur einige)

3.2.1 areaPoints

- Liste von Punkten (Koordinaten) einer zusammenhängenden Bildregion
- Besitzt Methoden zur Erzeugung der areaPoints aus channel8-Bildern (mit nur einer zusammenhängenden Region), Konturlisten, ….. (siehe Dokumentation).
- Besitzt Methode zur Umwandlung von areaPoints in ein channel8-Bild.
- Klasse *objectsFromMask* erzeugt aus einem binarisierten Bild mit beliebig vielen Bildregionen eine Liste von areaPoint-Listen.

```
-- 00000000011111111112
-- 01234567890123456789
00 --------------------
01 --------------------
02 --------------------
03 --------****--------
04 -------******-------
05 -------******-------
06 ------********------
07 -------******-------
08 --------------------
09 --------------------
10 --------------------
```

Liste der areaPoints

```
(8,3),(9,3),(10,3),(11,3),
(7,4),(8,4) ……  (13,7)
```

3.3 Bilder (nur einige)

3.3.1 image

Zweck: Bilddatenformat für RGB-Bilder

Wichtige Methoden: (siehe auch LTILib-Dokumentation)

`image()`
Konstruktor, legt Bild der Größe 0x0 an.

`image(const int &rows, const int &cols )`
Konstruktor, legt Bild der Größe `rows`x`cols` an.

`rows()`
liefert die Spaltenanzahl des image

`columns()`
liefert die Zeilenanzahl des image

`resize(const int &newRows, const int &newCols)`
Verändert die Bildgröße auf die angegebenen Werte.
Der alte Bildinhalt wird oben-links ausgerichtet.

`castFrom(const channel8 &other )`
Konvertiert ein `channel8`-Bild nach `image`.

Beispiel (Programmfragment)

```
// Image-Bilder anlegen
image Img1;                  // leeres Bild
image Img2(512, 512);        // Bild der Größe 512 x 512

// Instanziieren eines Bild-Loaders
loadBMP   myLoader;

// Instanziieren der Bild-Viewer
viewer    ViewerOriginalbild, ViewerKopie;

// lade Image-Bild
myLoader.load("Numbers.bmp", Img1);

// Größe von Img2 auf die Größe von Img1 setzen
x_Size = Img1.rows(); y_Size = Img1.columns();
Img2.resize(y_Size, x_Size);

// Bildpunkte von Img1 nach Img2 kopieren
for(y=0; y<y_Size; ++y)
     for(x=0; x<x_Size; ++x)
          Img2[y][x] = Img1[y][x];  // Anm: Datentyp: rgbPixel

// Bilder anzeigen
ViewerOriginalbild.show(Img1);
ViewerKopie.show(Img2);
```

3.3.2 channel8

Zweck: Bilddatenformat für Grauwertbilder (256 Grauwerte)

Wichtige Methoden: (siehe auch LTILib-Dokumentation)

`channel8()`
Konstruktor, legt Bild der Größe 0x0 an.

`channel8(const int &rows, const int &cols )`
Konstruktor, legt Bild der Größe `rows`x`cols` an.

`rows()`
liefert die Spaltenanzahl des image

`columns()`
liefert die Zeilenanzahl des image

`resize(const int &newRows, const int &newCols)`
Verändert die Bildgröße auf die angegebenen Werte.
Der alte Bildinhalt wird oben-links ausgerichtet.

`castFrom(const image &other )`
Konvertiert ein `image`-Bild nach `channel8` (Intensity-channel).

Beispiel (Programmfragment)

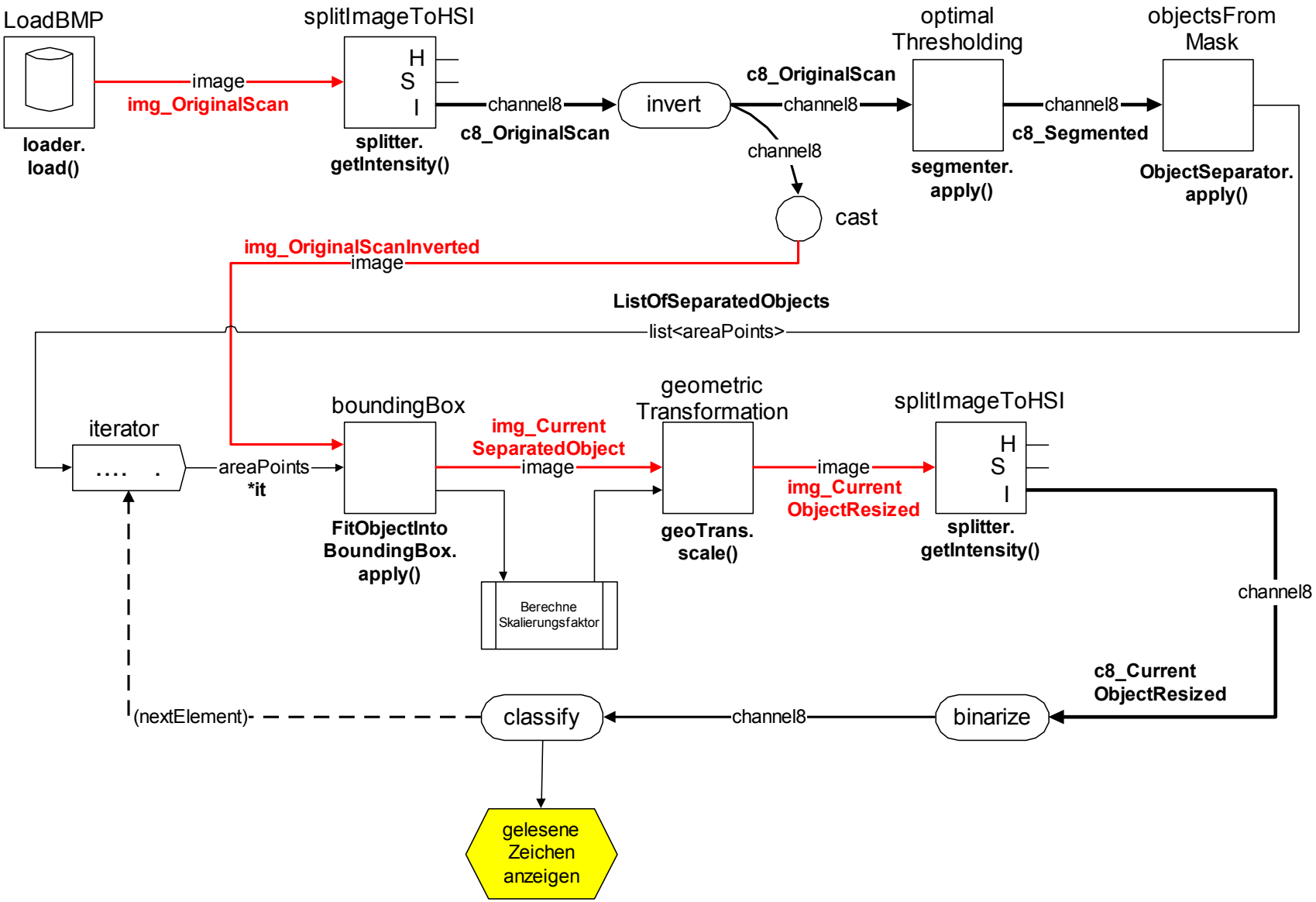
```
// Image-Bilder anlegen
image     Img1;              // leeres image-Bild anlegen
channel8  Pic1, Pic2;        // leeres channel8-Bilder anlegen

// Instanziieren eines Bild-Loaders und von 2 Bild-Viewern
loadBMP   myLoader;
viewer    ViewerOriginal, ViewerInvertiert;

// lade Image-Bild
myLoader.load("Numbers.bmp", Img1);

// Intensity-channel von Img1 nach Pic1 kopieren
Pic1.castFrom(Img1);

// Größe von Pic2 auf die Größe von Pic1 setzen
x_Size = Pic.rows(); y_Size = Pic.columns();
Pic2.resize(y_Size, x_Size);

// Berechne das Negativ von Pic1 → Pic2
for(y=0; y<y_Size; ++y)
     for(x=0; x<x_Size; ++x)
          Pic2[y][x] = 255 - Pic1[y][x];  // negieren

// Bilder anzeigen
ViewerOriginal.show(Pic1);
ViewerInvertiert.show(Pic2);
```

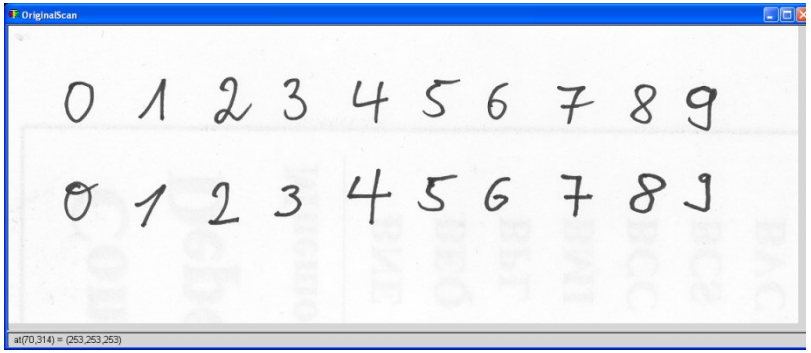
3.4 Bildverarbeitungsoperatoren

Bildverarbeitungsoperatoren (BV-Ops) sind als Klasse realisiert.

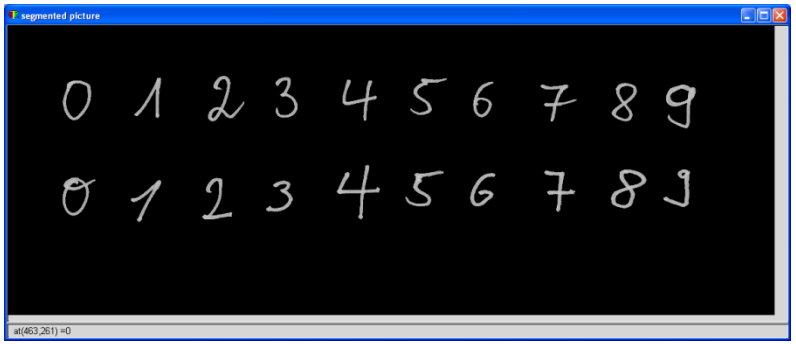
Jede Instanz eines BV-Ops besitzt einen eigenen Parametersatz, welcher sein Verhalten beeinflußt.

```
// Instanziiere CANNY-Operator (Kantenoperator)
cannyEdges  Canny;

// Instanziiere und initialisiere CANNY-Operator-Parameter
cannyEdges::parameters  Canny_Params = Canny.getParameters();

Canny_Params.thresholdMax =0.02;
Canny_Params.thresholdMin =0.99999;
Canny_Params.kernelSize   =15;
Canny_Params.variance     =0.7;

Canny.setParameters(Canny_Params);


// Filtere Bild „src“ und lege das Ergebnis in „dst“ ab
Canny.apply(src,dst);
```


LoadBMP
loader.
load()
image
img_OriginalScan
splitImageToHSI
H
S
I
splitter.
getIntensity()
channel8
c8_OriginalScan
invert
c8_OriginalScan
channel8
channel8
optimal
Thresholding
segmenter.
apply()
channel8
c8_Segmented
objectsFrom
Mask
ObjectSeparator.
apply()
channel8
cast
img_OriginalScanInverted
image
ListOfSeparatedObjects
list<areaPoints>
iterator
areaPoints
*it
boundingBox
FitObjectInto
BoundingBox.
apply()
img_Current
SeparatedObject
image
geometric
Transformation
geoTrans.
scale()
image
img_Current
ObjectResized
splitImageToHSI
H
S
I
splitter.
getIntensity()
Berechne
Skalierungsfaktor
channel8
c8_Current
ObjectResized
(nextElement)
classify
channel8
binarize
gelesene
Zeichen
anzeigen

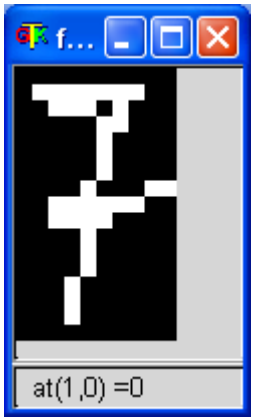


eingescanntes Bild



invertiertes und segmentiertes Bild

separiertes Zeichen



reskaliertes (10x16) und binarisiertes Zeichen

4. Integration von SNNS-Klassifikatoren (*.net)

SNNS:

- Trainiertes Netz abspeichern (*.net-Datei)
- mit `snns2c.exe` eine `*.c` und `*.h` Datei erzeugen (z.B. *numbers.c* + *numbers.h*)
- in beiden Dateien den Datentyp `float` durch `double` ersetzen
- in *numbers.h* der Funktionsdeklaration *extern “C“* voranstellen

```
extern "C" int numbers(double *in, double *out, int init);
```

im LTILib-Bildverarbeitungsprojekt

- *numbers.c* und *numbers.h* dem Projekt hinzufügen
- numbers(…) erwartet die Adressen des Ein- und Ausgangsvektors (**`double`**).
  - Vor dem Aufruf von Numbers muss daher das reskalierte Zeichenbild als double-Vektor abgelegt werden (→ Typanpassung).
  - Analog muss aus dem Ausgangsvektor die Klasse extrahiert werden.