ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG
INSTITUT FÜR INFORMATIK
Lehrstuhl für Mustererkennung und Bildverarbeitung
Prof. Dr.-Ing. Hans Burkhardt

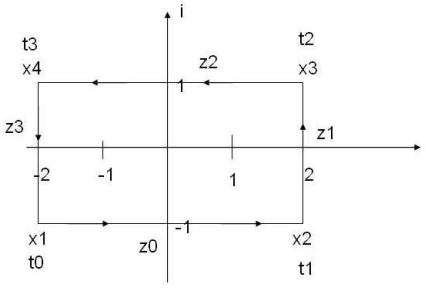
Georges-Köhler-Allee
Geb. 052, Zi 01-029
D-79110 Freiburg
Tel. 0761 - 203 - 8260

Übungen zur Vorlesung
Grundlagen der Bilderzeugung und Bildanalyse (Mustererkennung)
WS 05/06

Musterlösung 5

Aufgabe 5.1: Fourierkoeffizienten

1. Berechnung der Fourierkoeffizienten



$$c_n = \frac{T}{(2\pi n)^2} \sum_{k=0}^{N-1} (z_{k-1} - z_k) e^{-i\frac{2\pi n}{T} t_k}$$

$$c_0 = \frac{1}{2T} \sum_{k=0}^{N-1} (x_k + x_{k+1}) |x_{k+1} - x_k|$$

x_0	x_1	x_2	x_3
$-2-i$	$2-i$	$2+i$	$-2+i$

Polygonzug

wobei $z_i = \frac{x_{i+1}-x_i}{|x_{i+1}-x_i|}$ die normierte Differenz zwischen aufeinanderfolgenden Punkten ist (Einheitszeiger in Richtung der Polygonseiten, Umlaufrichtung mathematisch positiv).
Die Gesamtbogenlänge ergibt sich mit $T = 2 \cdot 4 + 2 \cdot 2 = 12$. Die aufsummierten Teilbogenlängen und normierten Differenzen ergeben sich durch Einsetzen zu

t_0	t_1	t_2	t_3	T
0	4	6	10	12

$\frac{t_0}{T}$	$\frac{t_1}{T}$	$\frac{t_2}{T}$	$\frac{t_3}{T}$
0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{5}{6}$

z_0	z_1	z_2	z_3
1	i	-1	$-i$

Berechnung von c_0:

$$c_0 = \frac{1}{2 \cdot 12} \cdot (-2 \cdot i \cdot 4 + 4 \cdot 2 + 2 \cdot i \cdot 4 + (-4) \cdot 2) = 0$$

Das ist der Konturschwerpunkt. Er ist im allgemeinen nicht gleich dem Flächenschwerpunkt.

Berechnung von c_n:

$$\begin{aligned} c_n &= \tfrac{12}{4(\pi n)^2}\left[(z_3-z_0)e^{-i\frac{2\pi n}{12}0}+(z_0-z_1)e^{-i\frac{2\pi n}{12}4}+(z_1-z_2)e^{-i\frac{2\pi n}{12}6}+(z_2-z_3)e^{-i\frac{2\pi n}{12}6}\right] \\ &= \tfrac{3}{(\pi n)^2}\left[(-i-1)+(1-i)e^{-i\frac{2\pi n}{3}}+(i+1)e^{-i\cdot\pi n}+(-1+i)e^{-i\frac{\pi n}{3}5}\right] \\ &= \frac{3}{(\pi n)^2}\left[(-i-1)+(1-i)e^{-i\frac{2\pi n}{3}}+(i+1)\cdot(-1)^n+(-1+i)e^{-i\frac{2\pi n}{3}}\cdot(-1)^n\right]] \end{aligned}$$

Wir können nun zwei Fälle unterscheiden:

1. n ist gerade: $c_n = 0$
2. n ist ungerade: $c_n = \frac{6}{(\pi\cdot n)^2}\left[(1-i)\cdot e^{-i\frac{2\pi n}{3}}-(i+1)\right]$

2. Durch die Aufpunktverschiebung (auf $\mathbf{x}_2$) erhalten wir eine neue Kontur

$$y(t) = x(t+4)$$

Dadurch ergeben sich auch andere Fourierkoeffizienten c'_n bzgl. y.

Für c'_n gilt

$$\begin{aligned} c'_n &= \frac{1}{T}\int_0^T y(t)e^{-i\frac{2\pi}{T}nt}dt = \frac{1}{T}\int_0^T x(t+4)e^{-i\frac{2\pi}{T}nt}dt = \frac{1}{T}\int_4^{T+4} x(t)e^{-i\frac{2\pi}{T}n(t-4)}dt \\ &= \frac{1}{T}\int_4^{T+4} x(t)e^{-i\frac{2\pi}{T}nt}e^{i4\frac{2\pi n}{T}}dt = e^{i4\frac{2\pi n}{T}}\frac{1}{T}\int_4^{T+4} x(t)e^{-i\frac{2\pi}{T}nt}dt \\ &= e^{i\frac{2\pi n}{3}}\underbrace{\frac{1}{T}\int_0^T x(t)e^{-j\frac{2\pi}{T}nt}dt}_{c_n} \\ c'_n &= e^{i\frac{2\pi n}{3}}c_n \end{aligned}$$

Damit lässt sich die Berechnung von c'_n auf c_n zurückführen.

Bei diesem Rechteck liegt eine Rotationsymmetrie von Grad 2, Achsensymmetrie bzgl. Imaginärachse und Achsensymmetrie bzgl. Realachse vor.

Die Rotationssymmetrie vom Grad 2 kann man daran erkennen, dass die geraden Koeffizienten Null sind, d.h. $c_n = 0$ für $n \neq 1 \pm 2k, k \in N$

Die Achsensymmetrie kann man folgendenmaßen von den Koeffizienten herleiten. Man sieht, dass durch Multiplikation mit Faktor $e^{i\frac{2\pi n}{6}}$ (dies bedeutet eine Aufpunktverschiebung in den Punkt $(0,-1)^T$), alle c_n rein imaginär werden. Das entspricht in Originalraum $x(-t) = -x(t)^*$ wobei * die komplexe Konjugation bezeichnet. Es ist zu sehen dass $x(-t) = -x(t)^*$ die Definition von Achsensymmetrie bzgl. imaginäre Achse ist.

Achsensymmetrie bzgl. reelle Achse kann ähnlich durch Multiplikation mit Faktor $e^{i\frac{-2\pi n}{12}}$ (Aufpunktverschiebung in den Punkt $(-2,0)^T$) hergeleitet werden, wobei alle c_n rein reall werden. Dies entspricht $x(-t) = x(t)^*$, was die Definition der Achsensymmetrie. bzgl. reelle Achse ist. Dies kann auch experimentell mit dem Scilab program `computeFc.sci` verifiziert werden, in dem man den Aufpunkt entsprechend verschiebt.

3. Das y-Rechteck entsteht durch Rotation um $\Phi = \frac{\pi}{2}$ und Translation um $z = (1,2)^T$ des x-Rechtecks. Außerdem ist der neue Aufpunkt verschoben mit $t_0 = -2$. Die Auswirkungen der Ahnlichkeitstransformation auf die Fourierkoeffizienten ergeben sich wegen der Linearitat der Fouriertransformation wie folgt. Deren Berechnung erfolgt unter der Annahme einer auf $T = 2 \cdot \pi$ normierten Bogenlange. (s. Folie 9 Kap. 4b)Da die Bogenlänge normiert ist gilt für den normierten Aufpunkt:

$$t_0' = \frac{t_0 \cdot 2 \cdot \pi}{T} = \frac{-2 \cdot 2 \cdot \pi}{12} = \frac{-\pi}{3}$$

$$c_0^y = c_0^x \cdot e^{i \cdot \Phi} + z = (1,2)^T$$

$$c_n^y = c_n^x \cdot e^{i \cdot (\Phi + n \cdot t_0')} = c_n^x \cdot e^{i(\frac{\pi}{2} - \frac{n\pi}{3})}$$

Aufgabe 5.2: Vektorielle und komplexe Fourierkoeffizienten

1. Betrachtet man die beiden Darstellungen eines Polygons als komplexes Konturmuster bzw. als vektorielle reelle Funktion, dann erkennt man folgenden Zusammenhang:

$$x(t) = u(t) + i \cdot v(t)$$

Um einen Zusammenhang zwischen den Fourierkoeffizienten der komplexen Darstellung und den U_k und V_k herzustellen, betrachten wir die Formel der FK's der komplexen Polygondarstellung:

$$\begin{aligned} c_k &= \frac{1}{T}\int_0^T x(t)e^{-i\omega kt}dt \qquad \omega = \frac{2\pi}{T} \\ &= \underbrace{\frac{1}{T}\int_0^T u(t)e^{-i\omega kt}dt}_{U_k} + i\,\underbrace{\frac{1}{T}\int_0^T v(t)e^{-i\omega kt}dt}_{V_k} \\ &= U_k + jV_k = (1,i)\begin{pmatrix} U_k \\ V_k \end{pmatrix} \end{aligned}$$

2. Eine weitere Gleichung werden wir erhalten, wenn wir in obige Gleichung $-k$ statt k einsetzen und die Beziehung zwischen U_k und U_{-k} bzw. zwischen V_k und V_{-k} berücksichtigen. Diese Beziehungen existieren, weil U_k bzw. V_k Fourierkoeffizienten von *reellen* Funktionen $u(t)$ bzw. $v(t)$ sind.
Um nun diese Beziehung zwischen U_k und U_{-k} herzuleiten, setzen wir $-k$ in obige Formel für U_k ein und erhalten:

$$U_{-k} = \frac{1}{T}\int_0^T u(t)e^{i\omega kt}dt = \left(\frac{1}{T}\int_0^T u(t)e^{-i\omega kt}dt\right)^*$$

Die gleiche Berechnung kann man für V_{-k} durchführen und erkennt:

$$U_{-k} = U_k^* \qquad V_{-k} = V_k^*$$

3. Die Formel aus Teil 1 ist zur Berechnung der U_k und V_k nicht ausreichend, da sich aus ihr nur eine Gleichung mit 2 Unbekannten ergibt. Nutzen wir aber zusätzlich die Erkenntnisse aus Teil 2, dann können wir, wie erwähnt, eine zweite Formel für c_{-k} angeben:

$$c_{-k} = (1, i) \begin{pmatrix} U_{-k} \\ V_{-k} \end{pmatrix} = (1, i) \begin{pmatrix} U_k^* \\ V_k^* \end{pmatrix}$$

Um nun aus U_k^* und V_k^* wieder deren nicht konjugierte Versionen zu erhalten, wird die gesamte Formel komplex konjugiert:

$$c_{-k}^* = (1, -i) \begin{pmatrix} U_k \\ V_k \end{pmatrix}$$

Nun haben wir zwei Gleichungen mit 2 Unbekannten:

$$\begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix} \begin{pmatrix} U_k \\ V_k \end{pmatrix} = \begin{pmatrix} c_k \\ c_{-k}^* \end{pmatrix}$$

Nach $\begin{pmatrix} U_k \\ V_k \end{pmatrix}$ aufgelöst ergibt sich:

$$\begin{pmatrix} U_k \\ V_k \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -i & i \end{pmatrix} \begin{pmatrix} c_k \\ c_{-k}^* \end{pmatrix}$$

Und in einzelnen Formeln ausgedrückt:

$$U_k = \frac{1}{2} \left(c_k + c_{-k}^* \right)$$

$$V_k = -\frac{i}{2} \left(c_k - c_{-k}^* \right)$$

Aufgabe 5.3: Programmieraufgabe: Fouriersynthese, Rotationssymmetrie

1.a) Es sollten mit Hilfe der Funktion

```
computeFC(n,Polygon)
```

die Fourierkoeffizienten für das regelmäßige Dreieck, das durch die 3 dritten komplexen Einheitswurzeln definiert ist, berechnet werden.

```
//Dreieck:
p3 = [%e^((2*%pi*%i*0)/3); ..
      %e^((2*%pi*%i*1)/3); ..
      %e^((2*%pi*%i*2)/3)];

//Berechnung der Fourierkoeffizienten
fc1d  = computeFc(1,p3);
fc7d  = computeFc(7,p3);
fc13d = computeFc(13,p3);
fc19d = computeFc(19,p3);
fc31d = computeFc(31,p3);
```

b) Für das Rechteck aus Aufgabe 5.1. werden die Fourierkoeffizienten entsprechend berechnet:

```
//Rechteck:
p4 = [-2-%i;2-%i;2+%i;-2+%i];

//Berechnung der Fourierkoeffizienten
fc1r  = computeFc(1,p4);
fc7r  = computeFc(7,p4);
fc13r = computeFc(13,p4);
fc19r = computeFc(19,p4);
fc31r = computeFc(31,p4);
```

2. Nun sollte eine Funktion zur Fouriersynthese implementiert werden, welche als Eingabe die Fourierkoeffizienten eines Objekts erwartet und die Fourierapproximation des Objekts zurückgibt.

```
function Papprox = Fsynthesis ( Fc )
// function Papprox = Fsynthesis ( Fc )
//
// Fouriersynthesis of a Polygon by use of the coefficients in Fc
//
// input:    Fc           vector of complex valued Fourier coefficients
//                        c_i, i ranging from -n,...,0,...,+n
//
// output:   Papprox      vector of 1000 complex values representing the
//                        approximate polygon.

// curve length
no = 10^3;
T = 0:1/no:1;
T = T'*2*%pi;

// index vector
N = -floor(length(Fc)/2):floor(length(Fc)/2);

// base functions
B = exp( %i.* T * N );

// Fouriersynthesis

Papprox = B * Fc;

endfunction;
```

Nun werden die Approximationen berechnet und anschließend geplottet.

a) Für das Dreieck:

```
s_1d  = Fsynthesis(fc1d);
s_7d  = Fsynthesis(fc7d);
s_13d = Fsynthesis(fc13d);
s_19d = Fsynthesis(fc19d);
s_31d = Fsynthesis(fc31d);
```

```
//Schließe Polygon
p3=[p3;p3(1)];

subplot(2,3,1)
xtitle('Original');
plot2d(real(p3),imag(p3));

subplot(2,3,2)
xtitle('N=1');
plot2d(real(s_1d),imag(s_1d));

subplot(2,3,3)
xtitle('N=7');
plot2d(real(s_7d),imag(s_7d));

subplot(2,3,4)
xtitle('N=13');
plot2d(real(s_13d),imag(s_13d));

subplot(2,3,5)
xtitle('N=19');
plot2d(real(s_19d),imag(s_19d));

subplot(2,3,6)
xtitle('N=31');
plot2d(real(s_31d),imag(s_31d));
```

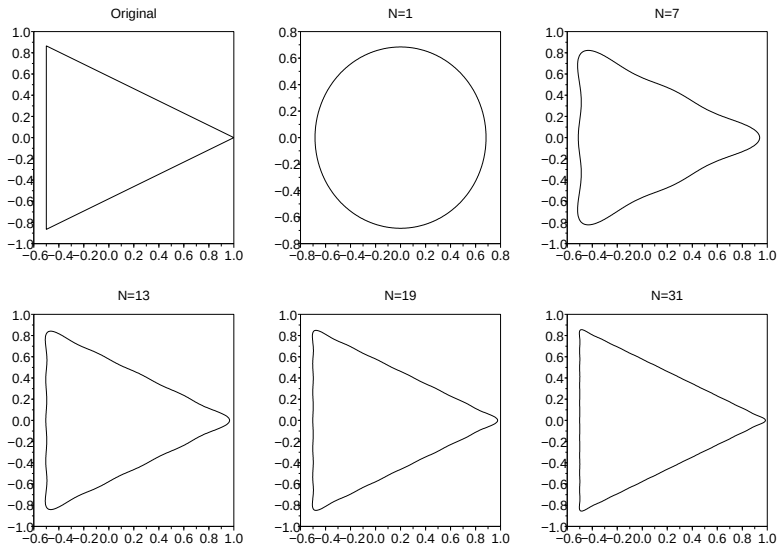


Abbildung 1: Approximationen mit verschiedenen Parametern

b) Für das Rechteck:

```
s_1r  = Fsynthesis(fc1r);
s_7r  = Fsynthesis(fc7r);
s_13r = Fsynthesis(fc13r);
s_19r = Fsynthesis(fc19r);
s_31r = Fsynthesis(fc31r);
```

```
//Schließe Polygon
p4=[p4;p4(1)];


subplot(2,3,1)
plot2d(0,0,axesflag=5,rect=[-3,-3,3,3]);
xtitle('Original');
plot2d(real(p4),imag(p4));

subplot(2,3,2)
xtitle('N=1');
plot2d(real(s_1r),imag(s_1r));

subplot(2,3,3)
xtitle('N=7');
plot2d(real(s_7r),imag(s_7r));

subplot(2,3,4)
xtitle('N=13');
plot2d(real(s_13r),imag(s_13r));

subplot(2,3,5)
xtitle('N=19');
plot2d(real(s_19r),imag(s_19r));

subplot(2,3,6)
xtitle('N=31');
plot2d(real(s_31r),imag(s_31r));
```

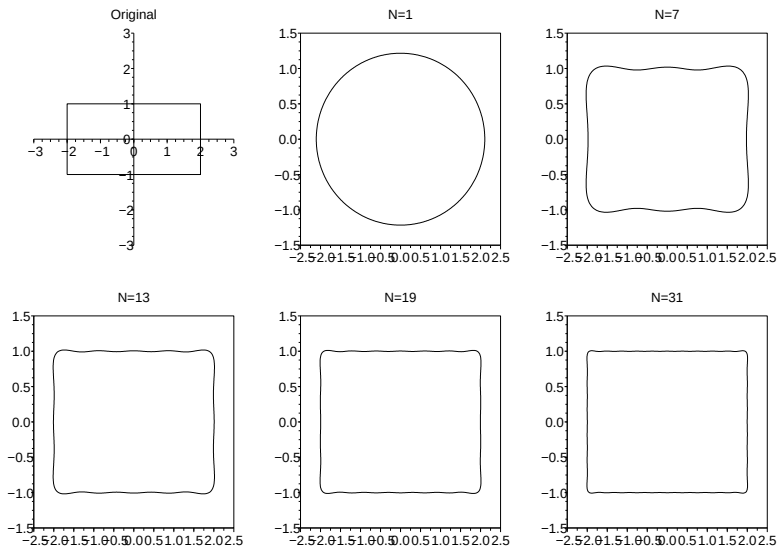


Abbildung 2: Approximationen mit verschiedenen Parametern

3 Implementierung der Funktion, welche anhand der Fourierkoeffizienten auf Rotationsymmetrie testet und den Grad zurückgibt.

```
function s = rotsym(Fc)
//---------------------------------------------------------------------
// function s = rotsym(Fc)
//
// Detection of rotational symmetry of an object that is represented
// by its Fourier coefficients
//
// input:    Fc       complex valued fourier coefficients in the order
//                    -c_n,...,c_0,...,c_n
//
//
// output:   s        estimated maximum rotational symmetry degree
//
//---------------------------------------------------------------------

  Fc = abs(Fc(:));
  Fc(ceil(length(Fc)/2)) = 0;

  // determine maximum Fc as reference point
  [FcMax,iMax] = max(Fc);
  tresh = 0.01 * FcMax

  // normalization of Fc by eliminating 1/n*n decay
  N = [-floor(length(Fc)/2):floor(length(Fc)/2)]';
  Fc = Fc .* N.^2;

  // for all possible symmetries sum over "should be 0 Fc".
  S(1) = 0;

  for s = 2:(length(Fc)/2)
    // Summation over relevant symmetries.
    n = 1:length(Fc);
    n_zero = find( modulo(abs(n-iMax),s) ~= 0);
    S(s) = sum(Fc(n_zero))/length(n_zero);
  end

  // decision
  N = (S < tresh)*1.0;

  // maximum symmetry
  [dummy,I] = sort(-N);
  s = I(length(I));

endfunction;
```

Das Programm verfährt wie folgt:
Bei einem vorliegenden Polygon ist die Anzahl der Punkte eine obere Schranke für den möglichen Symmetriegrad. Daher reichen doppelt so viele Fourierkoeffizienten aus, um diese maximale Symmetrie zu erkennen. Nach Berechnung dieser Fourierkoeffizienten werden diese auf ihre Absolutbeträge reduziert, und der Abfall der Beträge mit $1/n^2$ durch entsprechende Normierung kompensiert. Für jeden möglichen Symmetriegrad s werden die Absolutbeträge der Koeffizienten c_n mit $n \neq i_{max} + ks, k \in \mathbb{Z}$ summiert und durch die Anzahl dieser Koeffizienten dividiert. Da diese Koeffizienten bei vorliegender Symmetrie s alle verschwinden sollten, ist ein sehr kleiner Wert Indiz für vorliegen von Symmetrie s. Weil Symmetriegrad s bei einem Polygon automatisch alle Teiler von s als Symmetriegrad impliziert, wird am Ende das maximale der detektierten möglichen Symmetriegrade gewählt.

Die Ergebnisse auf den vorgegebenen Fourierkoeffizienten des Polygons lauten nun wie erwartet:

```
-->rotsym(fc31d)
 ans  = 3

-->rotsym(fc31r)
 ans  = 2
```