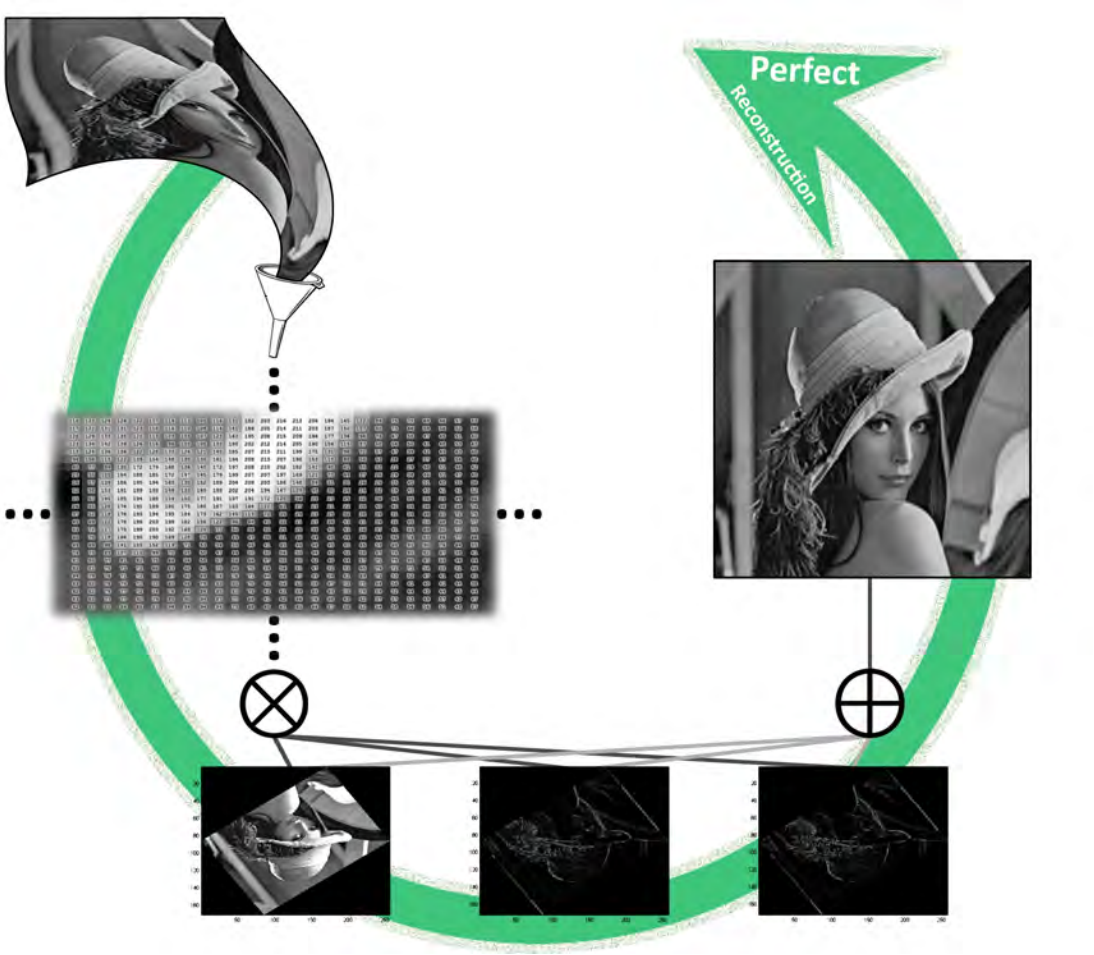
Bachelorarbeit

Filterbänke für Bildzerlegung

eingereicht bei: **Prof. Dr. Tomas Sauer**

am **20.07.2015**



Name, Vorname:	Wasmeier, Ewald
Matrikelnummer:	63660
Anschrift:	In der Point 1
	94554 Moos
Semesterzahl:	8
Studiengang:	Bachelor Informatik (ITS)
Telefonnummer:	0160/91835874
E-Mail:	wasmeier@fim.uni-passau.de

Inhaltsverzeichnis

1 Einleitung **4**

2 Hauptteil **5**

2.1 Funktionsdokumentationen der MatLab-Funktionen 5

2.1.1 DeleteLastAndFirstZeroRowsAndColumns.m 5

2.1.2 TransformMatrix.m 5

2.1.3 RelocateZeroElement.m 8

2.1.4 CalculateShiftVectorOfConvolution.m 8

2.1.5 ApplyShiftVectorToMatrix.m 8

2.1.6 Downsampling2D_1Dimension.m 10

2.1.7 Downsampling2D_DilationMatrix.m 11

2.1.8 Downsampling2D_DilationMatrix_ReturningAdditionalShiftVector.m 13

2.1.9 Upsampling2D_1Dimension 13

2.1.10 Upsampling2D_DilationMatrix 14

2.1.11 MultiLevelMultipleWaveletDecomposition2D.m 16

2.1.12 MultiLevelWaveletDecomposition2D.m 19

2.1.13 MultiLevelWaveletDecomposition2D_1Level.m 20

2.1.14 MultiLevelWaveletReconstruction2D.m 21

2.1.15 MultiLevelWaveletReconstruction2D_1Level.m 22

2.1.16 FilterDatabase.m 23

2.2 Funktionsdefinitionen der C-Funktionen 24

2.2.1 MexFunctions.h 24

2.2.2 WaveletDecompositionMex.c 29

2.2.3 MultipleWaveletDecompositionMex.c 30

2.3 Behandlung des Verschiebungsvektors 31

2.4 Mathematische Hintergründe 40

2.4.1 Smith-Faktorisierung 40

2.4.2 Quotientengruppen 41

2.4.3 Filter 48

2.4.4 Filterbänke 49

3 Anhang **57**

3.1 MatLab Erklärungen 57

3.1.1 Darstellung der Impulsantwort eines Filters 57

3.1.2 Smith-Faktorisierung 57

3.1.3 Downsampling und Upsampling 58

3.2 MEX Erklärungen 60
3.2.1 Speicherart von Matrizen 60

Abbildungsverzeichnis **62**

Literaturverzeichnis **63**

1 Einleitung

Bilder enthalten viele Informationen und Details. Oft ist man allerdings nur an bestimmten Einzelheiten eines Bildes interessiert. Dem Betrachter ist es ein Leichtes, allein diese Teilinformationen des ursprünglichen Bildes zu betrachten. Denn das Gehirn kann die vom Auge gelieferten Bilder ohne große Mühe nach den gewünschten Details filtern. Wenn es um Kantenerkennung, bestimmte Farbbereiche oder das Registrieren von ganzen Objekten im Bild geht, erbringt das menschliche Gehirn wahre Meisterleistungen.
Ein Computer steht hier vor sichtlich mehr Hindernissen. Denn digital besteht ein Bild nun mal nur aus Nullen und Einsen, in welche man grundsätzlich sehr schwierig brauchbare Informationen interpretieren kann. Einen Lösungsweg für dieses Problem stellt die Filterung des Bildes mit vorgegebenen Filterdaten dar. Dafür verwendet man eine Filtermatrix, von der man zum Beispiel weiß, dass sie horizontale Kanten im Bild sichtbar macht. Diese wird über die Bildmatrix geschoben (Bild und Filter werden miteinander gefaltet), wodurch man ein neues Bild erhält, welches nur die horizontalen Kanteninformationen des Ausgangsbildes beinhaltet. Je mehr solcher Filtermatrizen man verwendet, desto detaillierter lässt sich das Bild zerlegen.
Im mathematischen Teil der Arbeit wird versucht, eine Reihe von Filtermatrizen zu finden, welche perfekte Rekonstruktion versprechen. Das bedeutet, dass die zerlegten Detailbilder durch erneute Faltung mit inversen Filtermatrizen zum ursprünglichen Bild zusammengefügt werden können. Im programmiertechnischen Teil werden Funktionen näher beleuchtet, mit welchen man Bilder mit MatLab zerlegen und wieder zusammenfügen kann. Dazu sind zwei Beispiel-Filterbänke implementiert. Zur Zerlegung eines Bildes können sowohl MatLab-Funktionen als auch auf der Programmiersprache *C* basierende MEX-Funktionen verwendet werden, da MatLab durch geschachtelte Schleifen in der Regel sehr langsam wird. Diese Zeitverzögerung soll in der *C*-Implementierung vermieden werden.

2 Hauptteil

2.1 Funktionsdokumentationen der MatLab-Funktionen

Die verwendeten Funktionen basieren zum Teil auf den zu [1] gehörigen Funktionen.

2.1.1 DeleteLastAndFirstZeroRowsAndColumns.m

```
function sM = DeleteLastAndFirstZeroRowsAndColumns(M)
```

Die Funktion **DeleteLastAndFirstZeroRowsAndColumns** entfernt Nullzeilen und Nullspalten zu Beginn einer Matrix und an ihrem Ende. Nullzeilen oder -spalten in der Mitte einer Matrix sind davon nicht betroffen.
Folgendes Beispiel illustriert die Funktionsweise der Funktion:
Anmerkung: In der Praxis könnte diese Matrix ein schräg einbeschriebenes Bild enthalten.

Sei $M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{7} & \mathbf{9} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{8} & \mathbf{2} & \mathbf{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{5} & \mathbf{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{7} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, dann ist $sM = \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{7} & \mathbf{9} & 0 & 0 \\ 0 & \mathbf{8} & \mathbf{2} & \mathbf{4} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{5} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{7} & 0 \end{pmatrix}$

2.1.2 TransformMatrix.m

```
function tM = TransformMatrix(M, T)
```

Die Funktion **TransformMatrix** erhält eine zu transformierende Matrix M und eine Transformationsmatrix T. Letztere beinhaltet die Transformationsvorschrift. Die Funktion berechnet daraus nun die Umformung der Matrix M bezüglich der in T enthaltenen Vorschriften und gibt die so berechnete transformierte Matrix tM zurück.
Anhand des folgenden Beispiels wird die genaue Vorgehensweise dieser Transformation deutlich:
Seien $M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ und $T = \begin{pmatrix} 0 & 1 \\ 1 & -4 \end{pmatrix}$.

Anmerkung: T wurde nicht beliebig gewählt, sondern ist die Inverse der linken regulären

Matrix der Smith-Faktorisierung der Matrix $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$. Dies ist die Dilatationsmatrix einer in dieser Ausarbeitung oft verwendeten Filterbank.
Es gilt: $\Xi = U \cdot D \cdot V = \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 \\ -1 & -3 \end{pmatrix}$ und $T = U^{-1}$.

Die Funktion berechnet aus der Matrix M und der Transformationsmatrix T untenstehendes Ergebnis. Um die Form der berechneten Matrix tM genauer zu verstehen, ist in Abbildung 2.1 die Verformung des halboffenen Einheitsquadrats $[0, 1)^2$ in $\mathbb{Z}^2$ dargestellt. Die einzelnen Transformationsschritte findet man in Abbildung 2.2.

```
>> tM = TransformMatrix(M, T)

tM =

    12     0     0
    11     0     0
    10     0     0
     9     0     0
     0     8     0
     0     7     0
     0     6     0
     0     5     0
     0     0     4
     0     0     3
     0     0     2
     0     0     1
```

Bemerkung: Die Einträge auf der Diagonalen der Transformationsmatrix sorgen für eine X- bzw. Y-Verzerrung und die Nichtdiagonalelemente rufen Spiegelungen bzw. Drehungen hervor. Bei der Verzerrung wird die neue Matrix mit Nullen aufgefüllt, wo dies notwendig ist.
Bemerkung: Nach der Transformation der Matrix werden etwaige Nullzeilen oder -spalten am Anfang bzw. am Ende der Matrix gelöscht (siehe Abschnitt 2.1.1).

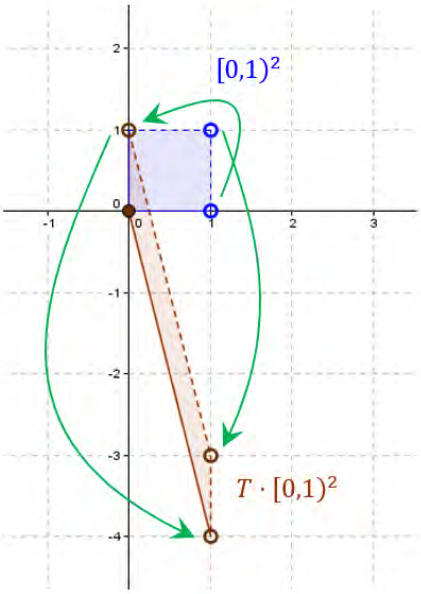


Abbildung 2.1: Die Transformationsmatrix T verzerrt das halboffene Einheitsquadrat $[0,1)^2$ zu einem Parallelogramm. Dieser Effekt ist vergleichbar mit der Wirkung der Transformationsmatrix auf die Matrix M.

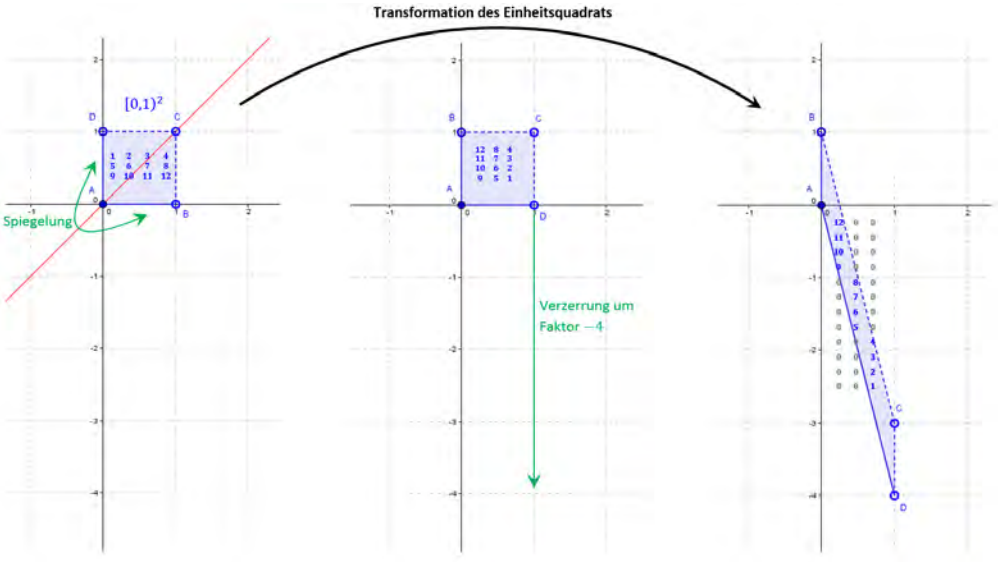


Abbildung 2.2: Die Transformationsmatrix T verzerrt das halboffene Einheitsquadrat $[0,1)^2$ zu einem Parallelogramm. Die im Einheitsquadrat abgebildeten Matrixeinträge sollen die Verzerrung und Spiegelung der Matrix M durch die Transformationsmatrix T verdeutlichen.

2.1.3 RelocateZeroElement.m

```
function M = RelocateZeroElement(M, sV)
```

Die Funktion **RelocateZeroElement** erhält eine zu verschiebende Matrix M und einen Verschiebungsvektor sV. Dieser Vektor wird direkt auf die Matrix angewendet, wodurch entweder Nullzeilen und -spalten zu Beginn der Matrix eingefügt werden (Einträge des Verschiebungsvektors sind positiv) oder Zeilen und Spalten zu Beginn der Matrix gelöscht werden (Einträge des Verschiebungsvektors sind negativ). Die Funktion wird von der MatLabfunktion **CalculateShiftVectorOfConvolution** aufgerufen und wird in deren Abschnitt 2.1.4 unter Angabe eines Beispiels genauer beschrieben.
Wird die Funktion nach der Faltung eines Bildes mit einer Matrix aufgerufen, dann muss das gefaltete Bild in der Regel um den Verschiebungsvektor der Filtermatrix zurückverschoben werden. Der Verschiebungsvektor entspricht einem Repräsentanten der Dilatationsmatrix der Filterbank.
Für die genaue Vorgehensweise zur Berechnung der Matrixrepräsentanten einer Dilatationsmatrix siehe Abschnitt 2.4.2.

2.1.4 CalculateShiftVectorOfConvolution.m

```
function sV = CalculateShiftVectorOfConvolution(M)
```

Die Funktion **CalculateShiftVectorOfConvolution** erhält eine Matrix M und berechnet die Position ihres ersten Eintrags, welcher ungleich Null ist. Sie wird benötigt, da sich der Verschiebungsvektor bei der Faltung immer auf den ersten Matrixeintrag ungleich Null bezieht.

Für $M = \begin{pmatrix} 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 & 6 \\ 0 & 7 & 8 & 9 & 10 \end{pmatrix}$ ergibt sich $sV = (2, 1)$.

Begründung: Das erste Element ungleich Null in der Matrix M ist um zwei Zeilen nach unten und um eine Spalte nach rechts verschoben.
Anmerkung: Die Nummerierung der Matrixelemente beginnt bei dieser Funktion sowohl zeilenweise als auch spaltenweise bei 0 anstatt wie in MatLab üblich bei 1. Dies liegt daran, dass letztendlich nicht die Position des ersten Elements ungleich Null sondern dessen Verschiebungsvektor zum eigentlichen ersten Element der Matrix betrachtet wird. Denn dadurch kann die Matrix nach dem Upsampling wieder korrekt ausgerichtet werden.

2.1.5 ApplyShiftVectorToMatrix.m

```
function sM = ApplyShiftVectorToMatrix(M, sV)
```

Die Funktion **ApplyShiftVectorToMatrix** erhält eine zu verschiebende Matrix M und einen Verschiebungsvektor sV, welcher auf diese Matrix angewendet werden soll. Der Verschiebungsvektor besitzt zwei Einträge, die den Versatz in X- und Y-Richtung

darstellen. Diese Funktion löscht nun entweder beginnende Zeilen und Spalten in der Matrix oder fügt Nullzeilen und -spalten hinzu, um dem Verschiebungsvektor gerecht zu werden.
Da sich der angegebene Verschiebungsvektor auf das erste Element der Bildmatrix, welches ungleich Null ist, bezieht, muss zuerst dessen Position in der Matrix berechnet werden. Anschließend wird die berechnete Position vom Verschiebungsvektor abgezogen, sodass die linke obere Ecke der Matrix korrekt platziert wird. Für die eigentliche Verschiebung der Bildmatrix wird die Funktion **RelocateZeroElement** aufgerufen.

Folgendes Beispiel macht die Vorgehensweise der Funktion deutlich:

Sei $M = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \end{pmatrix}$ und $sV = \begin{pmatrix} 3 & 2 \end{pmatrix}$.

Die Position des ersten Elements ungleich Null ist $pos = (1, 0)$. Diese wird mit Hilfe der Funktion **CalculateShiftVectorOfConvolution** (siehe Abschnitt 2.1.4) berechnet und beginnt im Gegensatz zur gängigen MatLab-Nummerierung bei 0 und nicht bei 1. Dies hat den einfachen Grund, dass der berechnete Versatz somit direkt von dem Verschiebungsvektor subtrahiert werden kann:
Der neue Verschiebungsvektor ergibt sich aus:
$sVnew = sV - pos = \begin{pmatrix} 3 & 2 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 2 \end{pmatrix}$
Dieser Vektor wird nun auf die Matrix M angewendet, was in Abbildung 2.3 verdeutlicht wird.

$$sM = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 5 & 6 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 12 & 13 & 14 \end{pmatrix}$$

Abbildung 2.3: Der berechnete Verschiebungsvektor $sVnew$ wird auf die Matrix M angewendet. Dabei werden Nullzeilen und -spalten zu Beginn der Matrix eingefügt.

Das nachfolgende Beispiel verdeutlicht die Wirkung von **ApplyShiftVectorToMatrix** auf eine Matrix, wenn ein negativwertiger Verschiebungsvektor angegeben wird:

Sei dazu $M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ und $sV = \begin{pmatrix} -1 & -2 \end{pmatrix}$.

Das Resultat der Funktion wird in Abbildung 2.4 dargestellt. Da das erste Element der angegebenen Matrix auch dem ersten Element ungleich Null entspricht, muss der als

Parameter übergebene Verschiebungsvektor sV nicht neu berechnet werden.

$$sM = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 11 & 12 \end{pmatrix}$$

Abbildung 2.4: Ist der berechnete Verschiebungsvektor negativ, so werden Zeilen und Spalten zu Beginn der Matrix gelöscht.

Für die genaue Beschreibung der Behandlung des Verschiebungsvektors der Matrizen während des Samplings und der Faltung siehe Abschnitt 2.3.

2.1.6 Downsampling2D_1Dimension.m

```
function dM = Downsampling2D_1Dimension(M, dim, fac)
```

Die Funktion **Downsampling2D_1Dimension** führt auf der Matrix M ein Downsampling um den Faktor fac entlang der Dimension dim (mögliche Eingaben: $'X'$, $'Y'$) durch. Folgendes Beispiel verdeutlicht die Vorgehensweise dabei:

Sei dazu $M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \end{pmatrix}$ und $facX = 3$, $facY = 2$.

Der Aufruf

```
dM = Downsampling2D_1Dimension(M, 'X', facX)
```

behält nur jede $facX$-te Zeile (hier also jede dritte Zeile) und streicht alle anderen Zeilen der Matrix.
Der Aufruf

```
dM = Downsampling2D_1Dimension(M, 'Y', facY)
```

würde jede $facY$-te Spalte (hier also jede zweite Spalte) beibehalten und alle anderen Spalten löschen.

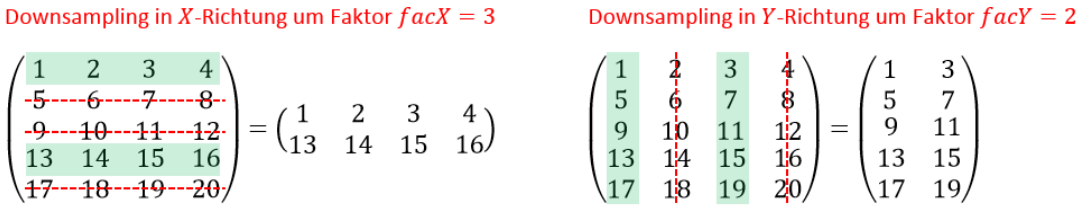


Abbildung 2.5: Die Matrix M wird links in X-Richtung um Faktor 3 und rechts in Y-Richtung um Faktor 2 einem Downsampling unterzogen.

Anmerkung: Beim Aufruf der Funktion mit negativen Samplingfaktoren oder Null wird eine Fehlermeldung geworfen, da dies keinen Sinn ergibt. Die Angabe von Eins als Samplingfaktor ändert an der Matrix nichts.
Theoretisch könnte man negative Faktoren als Upsampling interpretieren, dafür gibt es allerdings eine andere Methode (siehe 2.1.9).

2.1.7 Downsampling2D_DilationMatrix.m

```
function dM = Downsampling2D_DilationMatrix(M, S)
```

Die Funktion **Downsampling2D_DilationMatrix** erhält eine abzutastende Matrix M und eine Streckungsmatrix S welche die Informationen zum Downsampling beinhaltet. Dabei hängt der Prozess des Downsampling direkt von der Gestalt der Streckungsmatrix ab. Ist diese etwa in Diagonalgestalt gegeben, so werden nacheinander eindimensionale Downsampling-Verfahren in Richtung der beiden Dimensionen der Matrix mit den Diagonaleinträgen von S als Samplingfaktoren durchgeführt (siehe 2.1.6).

Besitzt die Streckungsmatrix S keine Diagonalgestalt, so ist das Vorgehen etwas komplexer. Folgende Auflistung beschreibt die Vorgehensweise der Funktion:

1. Zerlege die Matrix S mit Hilfe der Smith-Faktorisierung.
2. Lese die Skalierungsfaktoren für das Downsampling in Richtung der beiden Dimensionen von der Diagonalen der Smith-Normalform ab.
3. Transformiere die eigentliche Matrix M unter Verwendung der Inversen der linken Smith-Matrix mittels **TransformMatrix.m** (siehe 2.1.2).
4. Führe jeweils eindimensionale Downsampling-Prozesse in Richtung der beiden Dimensionen der transformierten Matrix mit den bereits abgelesenen Skalierungsfaktoren durch.
5. Transformiere die downgesampelte Matrix unter Verwendung der Inversen der rechten Smith-Matrix mittels **TansformMatrix.m** wieder zurück.

Im Folgenden wird der Prozess des Downsampling bezüglich einer Streckungsmatrix durch Angabe eines Beispiels näher beleuchtet.

Sei dazu $M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ und $S = \Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$.

Die Smith-Faktorisierung $S = U \cdot D \cdot V = \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 \\ -1 & -3 \end{pmatrix}$ ist bereits bekannt.

Zudem gilt: $U^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -4 \end{pmatrix}$ und $V^{-1} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}$.

Nun liest man von der Smith-Normalform D die Skalierungsfaktoren in Richtung der einzelnen Dimensionen ab:

- $facX = D[2,2] = 3$
- $facY = D[1,1] = 1$

Das bedeutet, nach der Transformation der Matrix darf von dieser nur noch jede dritte Zeile behalten werden. An der Spaltenzahl ändert sich aufgrund des Faktors 1 nichts.
Zuerst muss die Matrix M allerdings transformiert werden. Diese Arbeit wurde bereits in Abschnitt 2.1.2 erledigt und wir erhalten als transformierte Matrix tM, welche mit Faktor 3 in X-Richtung einem Downsampling unterzogen wird und schließlich unter Verwendung von V^{-1} zu M' rücktransformiert wird:

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}, tM = \begin{pmatrix} 12 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 7 & 0 \\ 0 & 6 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 3 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}, dtM = \begin{pmatrix} 12 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 3 \end{pmatrix} \text{ und } M' = \begin{pmatrix} 0 & 0 & 12 \\ 9 & 6 & 3 \end{pmatrix}$$

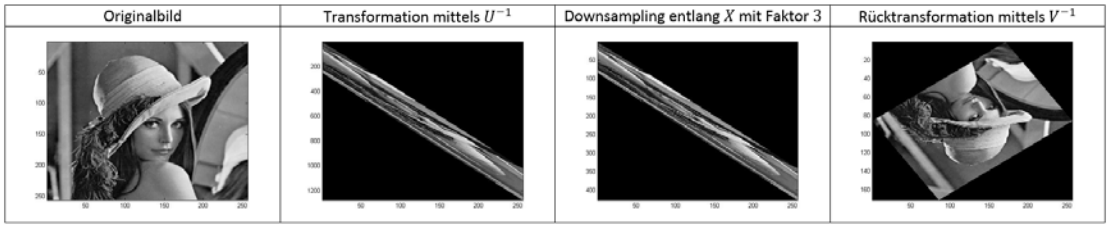


Abbildung 2.6: Das Testbild 'Lena' wird bezüglich der Matrix $S = \Xi$ einem Downsampling unterzogen.

Anmerkung: In der Regel findet auch ein Downsampling in Y-Richtung statt. Bei der Dilatationsmatrix Ξ wird dies allerdings immer übersprungen, da ein Sampling mit Faktor 1 nichts an der Eingabematrix verändert.

2.1.8 Downsampling2D_DilationMatrix_ReturningAdditionalShiftVector.m

```
function [dM, sV] =
   Downsampling2D_DilationMatrix_ReturningAdditionalShiftVector(M, S)
```

Die Funktion **Downsampling2D_DilationMatrix_ReturningAdditionalShiftVector** ergänzt die vorangehende Funktion um die Erweiterung der Berechnung des durch das Downsampling hervorgerufenen Verschiebungsvektors. Dabei wird zuerst eine Matrix mit der gleichen Gestalt wie die übergebene Bildmatrix M erzeugt, welche aufsteigende, eindeutige Elemente enthält. Diese Positionsmatrix wird genau wie die Bildmatrix dem Downsampling und den Transformationen unterzogen. Aufgrund der Eindeutigkeit ihrer Elemente kann im Nachhinein der Verschiebungsvektor berechnet werden (nähere beispielhafte Erläuterungen dazu in Abschnitt 2.3).

2.1.9 Upsampling2D_1Dimension

```
function uM = Upsampling2D_1Dimension(M, dim, fac)
```

Die Funktion **Upsampling2D_1Dimension** führt auf der Matrix M ein Upsampling um den angegebenen Faktor fac entlang der Dimension dim (mögliche Eingaben: $'X'$, $'Y'$) durch. Folgendes Beispiel verdeutlicht die Vorgehensweise dabei:

Sei dazu $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ und $facX = 3$, $facY = 2$.

Der Aufruf

```
uM = Upsampling2D_1Dimension(M, 'X', facX)
```

fügt zwischen je zwei Zeilen jeweils $(facX - 1)$ Nullzeilen ein.
Der Aufruf

```
uM = Upsampling2D_1Dimension(M, 'Y', facY)
```

würde zwischen je zwei Spalten jeweils $(facY - 1)$ Nullspalten hinzufügen.

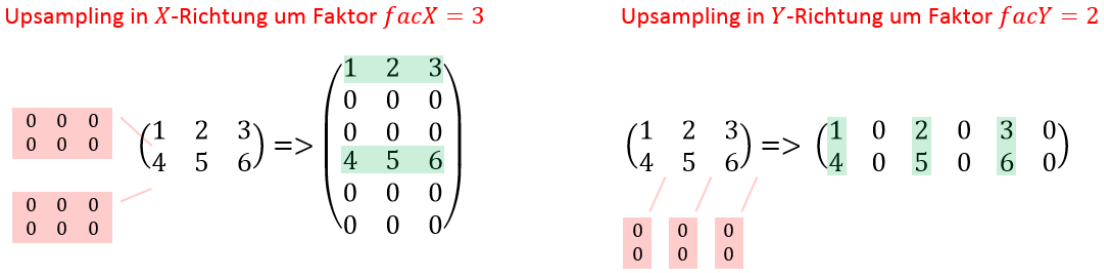


Abbildung 2.7: Die Matrix M wird links in X-Richtung um Faktor 3 und rechts in Y-Richtung um Faktor 2 einem Upsampling unterzogen.

2.1.10 Upsampling2D_DilationMatrix

```
function uM = Upsampling2D_DilationMatrix(M, S)
```

Die Funktion **Upsampling2D_DilationMatrix** nimmt als Parameter eine Matrix M und eine Dilatationsmatrix S an. Die Dilatationsmatrix enthält die Informationen über den Upsamplingvorgang. Tritt der Spezialfall ein, dass die Dilatationsmatrix S Diagonalgestalt hat, so werden nur nacheinander eindimensionale Upsampling Prozesse in Richtung der beiden Dimensionen gestartet (siehe Abschnitt 2.1.9).

Besitzt die Matrix S keine Diagonalgestalt, so wird beim Upsampling wie folgt vorgegangen:

1. Zerlege die Dilatationsmatrix S mit Hilfe der Smith-Faktorisierung.
2. Lese die Streckungsfaktoren für das Upsampling von der Diagonalen der Smith-Normalform ab.
3. Transformiere die Bildmatrix M unter der Verwendung der rechten regulären Matrix der Smith-Faktorisierung mittels **TransformMatrix.m** (siehe Abschnitt 2.1.2).
4. Führe jeweils eindimensionale Upsampling-Prozesse in Richtung der beiden Dimensionen der Matrix M mit den in Schritt 2 abgelesenen Skalierungsfaktoren durch.
5. Rücktransformiere die upgesampelte Matrix unter Verwendung der linken regulären Matrix der Smith-Faktorisierung mittels **TransformMatrix.m**.

Folgendes Beispiel soll das Downsampling aus Abschnitt 2.1.7 umkehren und die ursprüngliche Matrix wieder herstellen.

Sei dazu $M' = \begin{pmatrix} 0 & 0 & 12 \\ 9 & 6 & 3 \end{pmatrix}$ und $S = \Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$
Die Smith-Faktorisierung $S = U \cdot D \cdot V = \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 \\ -1 & -3 \end{pmatrix}$ ist bereits bekannt.
Wieder werden die Skalierungsfaktoren in beide Richtungen abgelesen:

- $facX = D[2,2] = 3$
- $facY = D[1,1] = 1$

Zwischen jede Zeile müssen also $(facX - 1) = 2$ Nullzeilen. Nullspalten müssen aufgrund des Faktors $facY = 1$ keine hinzugefügt werden. Vor dem Upsampling muss die Matrix M' allerdings erst unter Berücksichtigung der Transformationsmatrix U zu tM transformiert werden. Daraufhin wird sie einem Upsampling unterzogen, woraufhin man die Matrix utM erhält. Nach der Rücktransformation mit V ergibt sich die Ausgangsmatrix M. Hier fehlen allerdings einige Einträge, denn diese wurden der Matrix beim Downsampling entzogen.

$$M' = \begin{pmatrix} 0 & 0 & 12 \\ 9 & 6 & 3 \end{pmatrix}, \; tM = \begin{pmatrix} 12 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \; utM = \begin{pmatrix} 12 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; M = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \\ 9 & 0 & 0 & 12 \end{pmatrix}$$

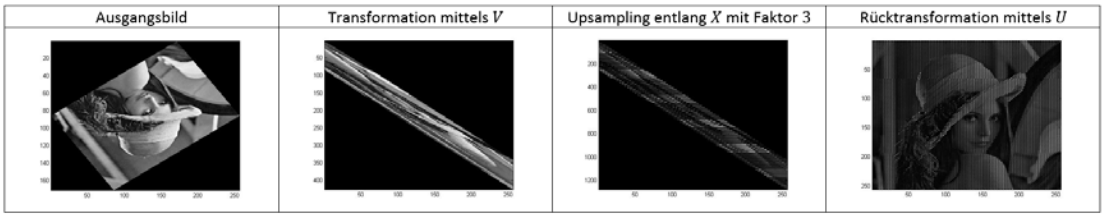


Abbildung 2.8: Das downgesampelte Testbild 'Lena' wird bezüglich der Matrix $S = \Xi$ einem Upsampling unterzogen. Man erkennt sehr schön, dass dem Resultat die Bildinformation, welche beim Downsampling entzogen wurde, fehlt. Bei der Rekonstruktion des Bildes würde man diese fehlende Informationen aus den Hochpassergebnissen beziehen.

2.1.11 MultiLevelMultipleWaveletDecomposition2D.m

```
function [CL, CH] =
   MultiLevelMultipleWaveletDecomposition2D(M, lvl, F1, F2)
```

Die Funktion **MultiLevelMultipleWaveletDecomposition2D** erhält eine Matrix M (welche üblicherweise ein Bild darstellt), zwei Filterbanknamen $F1$ und $F2$ und ein maximales Analyselevel lvl als Parameter. Für jedes Level bis hin zum maximalen Analyselevel wird für jede der beiden Filterbänke die Funktion **MultiLevelWaveletDecomposition2D_1Level** gestartet, wobei die aktuell zu filternde Matrix und der Name der benötigten Filterbank übergeben wird. Deren berechnete Filterergebnisse wiederum werden an der richtigen Stelle des Cell-Arrays CL für Tiefpass-Ergebnisse bzw. CH für Hochpass-Ergebnisse gespeichert. Jedes Zellenelement von CL und CH besteht wiederum aus einem (1×3)-Cell-Array, dessen erster Eintrag das Hochpass- bzw. Tiefpassergebnis speichert. Der zweite Eintrag enthält die Filterhistorie, d.h. eine Auflistung der Namen aller Filter, welche nacheinander auf das Originalbild angewendet wurden, um zum aktuellen Ergebnis zu kommen. Und der dritte und letzte Eintrag speichert schließlich noch den Verschiebungsvektor, welcher bei der Rekonstruktion auf die Bildinformationen vor der Summation angewendet werden muss.
Abbildung 2.9 zeigt die beiden Cell-Arrays zu einem Testbild. In Abbildung 2.10 wird der Zusammenhang der einzelnen gefilterten Bilder deutlich.

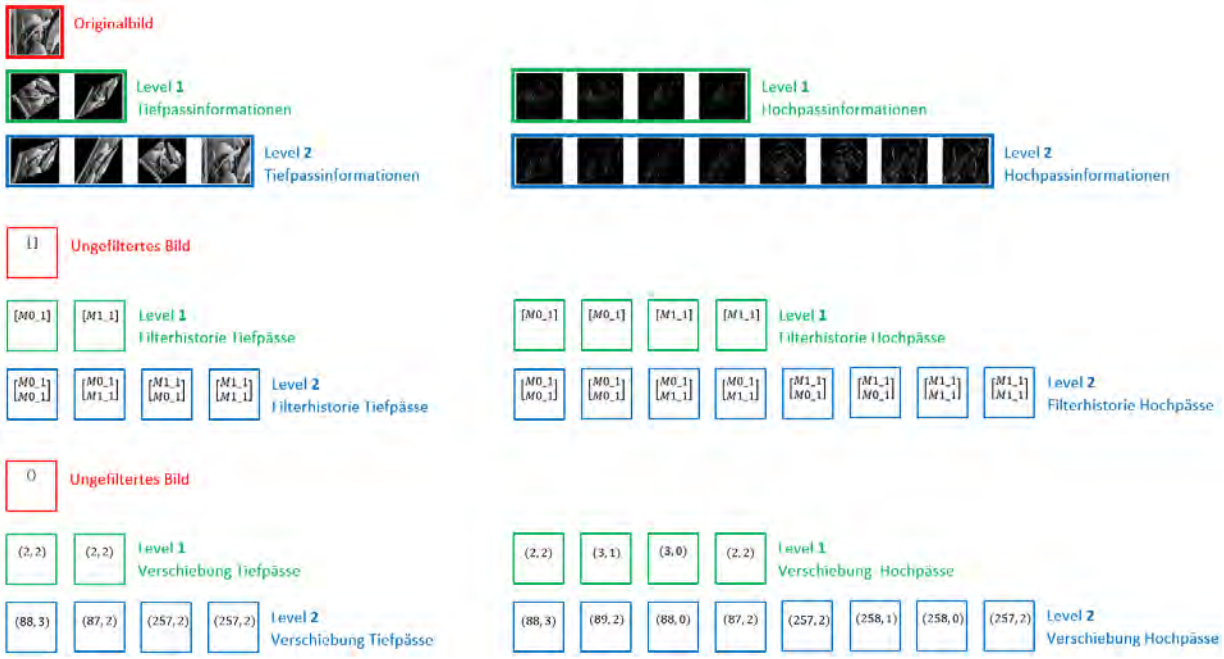


Abbildung 2.9: Das Testbild 'Lena' wird durch die zwei Filterbänke *Our Interpol M0_1* und *Our Interpol M1_1* zerlegt. Oben links sind die Tiefpass-Ergebnisse und oben rechts die Hochpass-Ergebnisse dargestellt. Eine Zeile entspricht jeweils einem Analyselevel. Für die Analyse wurde hier ein maximales Level von 2 gewählt. Im mittleren Teil der Abbildung wird deutlich, wie die Filterhistorie für die einzelnen Teilergebnisse gespeichert wird. Diejenigen Filterergebnisse mit derselben Filterhistorie aus dem Tiefpass-Array und dem Hochpass-Array gehören zusammen. Ganz unten schließlich sieht man die gespeicherten Verschiebungsvektoren, welche für die Rekonstruktion benötigt werden.

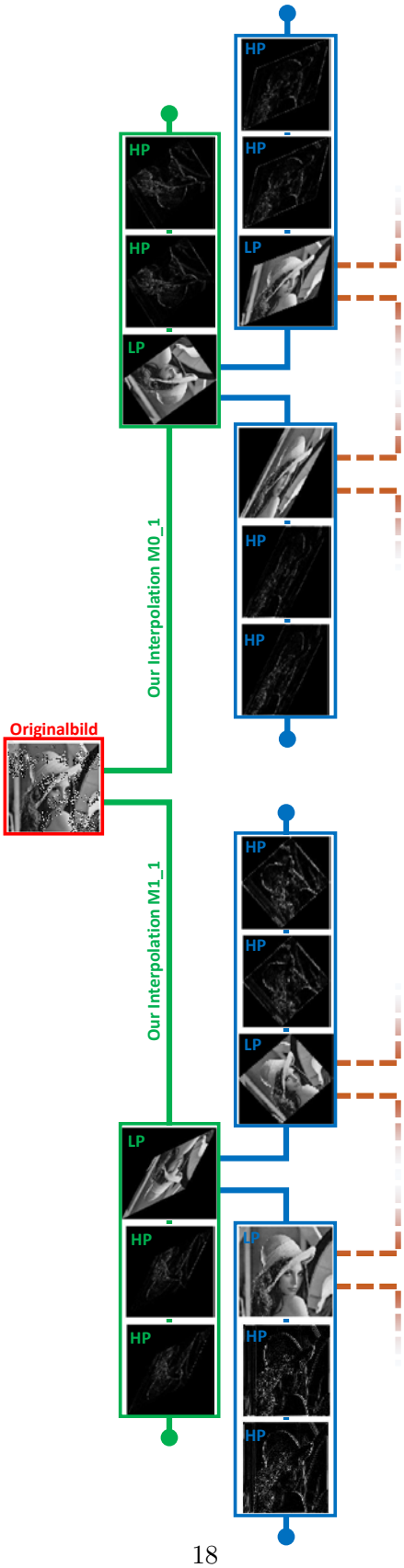


Abbildung 2.10: Das Testbild 'Lena' wird durch die zwei Filterbänke *Our Interpol M0_1* und *Our Interpol M1_1* zerlegt. Im Baum sind die Abhängigkeiten voneinander dargestellt. In jedem Level werden die Tiefpass-Ergebnisse erneut gefiltert. Filterergebnisse in der gleichen Spalte sind vom gleichen Level.

2.1.12 MultiLevelWaveletDecomposition2D.m

```
function [CL, CH] = MultiLevelWaveletDecomposition2D(M, lvl, F)
```

Die Funktion **MultiLevelWaveletDecomposition2D** erhält eine Matrix M (welche üblicherweise ein Bild darstellt), einen Filterbanknamen F und ein maximales Analyselevel lvl als Parameter. Für jedes Level bis hin zum maximalen Analyselevel wird die Funktion **MultiLevelWaveletDecomposition2D_1Level** gestartet, wobei die aktuell zu filternde Matrix und der Filterbankname F übergeben wird. Die berechneten Filterergebnisse wiederum werden an der richtigen Stelle des Cell-Arrays CL für Tiefpass-Ergebnisse bzw. CH für Hochpass-Ergebnisse gespeichert. Jedes Element von CL bzw. CH selbst besteht wiederum aus einem (1×3)-Cell-Array, wobei dessen erster Eintrag die gefilterte Matrix und der zweite Eintrag die Filterhistorie beinhaltet. Der dritte Eintrag beinhaltet den Verschiebungsvektor für die Rekonstruktion.
Im Prinzip wäre hier keine zusätzliche Speicherung des Filterbanknamens notwendig gewesen, da ja nur eine einzige Filterbank verwendet wurde. Um das System einheitlich zu gestalten wird allerdings eine Filterhistorie wie bei Abschnitt 2.1.11 angelegt. Dies erleichtert auch die Rekonstruktion der gefilterten Daten erheblich.
Abbildung 2.11 zeigt das Ergebnis einer 3-Level Analyse des bekannten Testbildes 'Lena'. Abbildung 2.12 wiederum verdeutlicht den Zusammenhang der einzelnen gefilterten Ergebnisse.

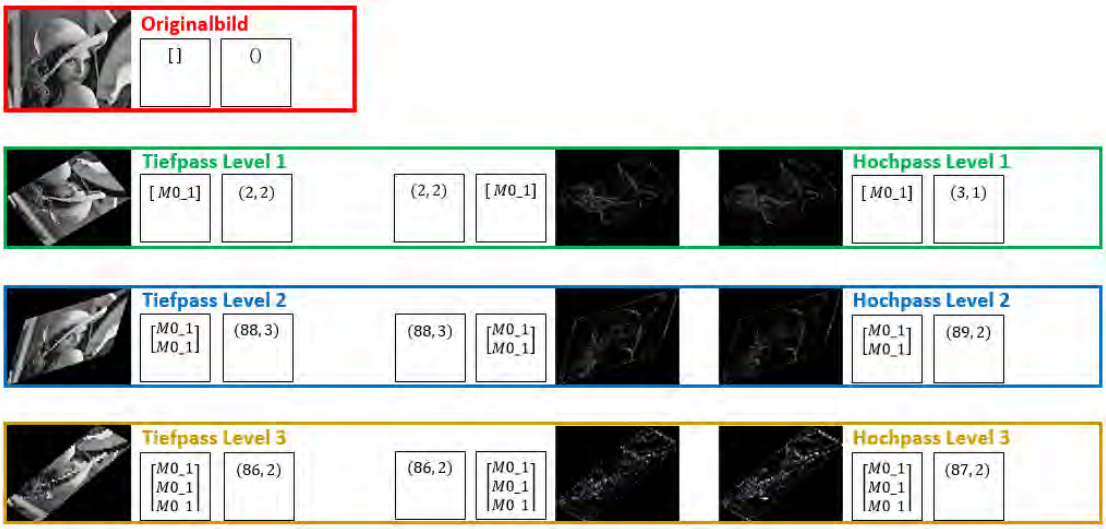


Abbildung 2.11: Das Testbild 'Lena' wird durch die Filterbank *Our Interpol M0_1* zerlegt. Links sind die Tiefpass-Ergebnisse und rechts die Hochpass-Ergebnisse dargestellt. Neben jedem Filterergebnis steht seine Filterhistorie, welche in diesem Fall trivial ist. Zudem ist der Verschiebungsvektor für jeden Tiefpass und Hochpass im Cell-Array gespeichert. Eine Zeile entspricht jeweils einem Analyselevel. Für die Analyse wurde hier ein maximales Level von 3 gewählt.

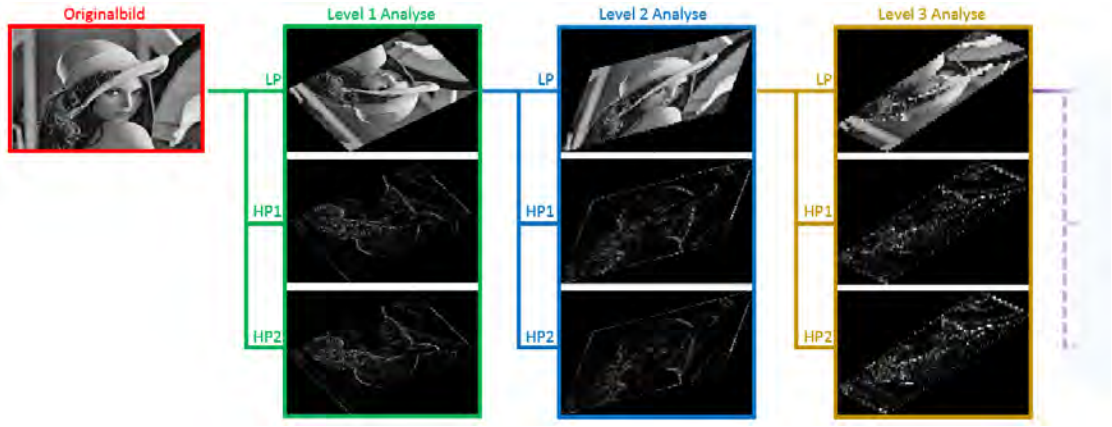


Abbildung 2.12: Das Testbild 'Lena' wird durch die Filterbank *Our Interpol M0_1* zerlegt. Im Baum sind die Abhängigkeiten voneinander dargestellt. In jedem Level werden die Tiefpass-Ergebnisse erneut gefiltert. Filterergebnisse in der gleichen Spalte sind vom gleichen Level.

2.1.13 MultiLevelWaveletDecomposition2D_1Level.m

```
function [l, lsV, h1, h1sV, h2, h2sV] =
   MultiLevelMultipleWaveletDecomposition2D\_1Level(M, F)
```

Die Funktion **MultiLevelWaveletDecomposition2D_1Level** erhält eine Matrix M und einen Filterbanknamen F. Sie zerlegt die übergebene Matrix M in Tiefpass- und Hochpassanteile und gibt diese zurück. Für jede Zerlegung wird ihr Verschiebungsvektor lsV, $h1sV$ oder $h2sV$ berechnet und zusätzlich zurückgegeben. Dieser kann dann für die Rekonstruktion gespeichert und verwendet werden. Die Funktion wird in der Regel von **MultiLevelMultipleWaveletDecomposition2D** bzw. **MultiLevelWaveletDecomposition2D** für jedes Analyselevel einmal aufgerufen.
Intern werden zuerst die Filterkoeffizienten aus der Filterdatenbank **FilterDatabase** aufgerufen. Danach wird die Matrix M mit den einzelnen Filtermatrizen gefaltet und im Anschluss einem Downsampling unterzogen. Während des gesamten Prozesses wird der Verschiebungsvektor aller drei Zerlegungen aktualisiert.
Abbildung 2.13 zeigt das Testbild 'Lena', welches einem Hochpassfilter unterzogen wird.

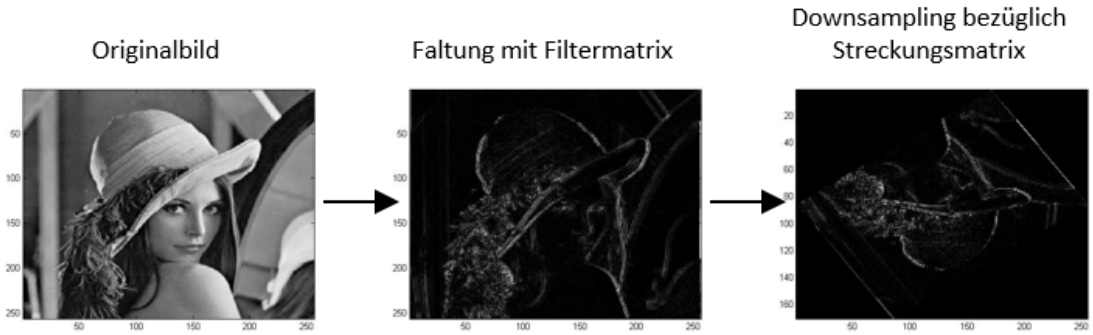


Abbildung 2.13: Das Testbild 'Lena' wird zuerst mit der Koeffizientenmatrix eines Hochpassfilters gefaltet und im Anschluss daran einem Downsampling unterzogen.

2.1.14 MultiLevelWaveletReconstruction2D.m

```
function [LR, HR] = MultiLevelWaveletReconstruction2D(lc, CL, CH)
```

Die Funktion **MultiLevelWaveletReconstruction2D** erhält ein Element aus einem Tiefpass-Cell-Array, das komplette Tiefpass-Cell-Array und das komplettes Hochpass-Cell-Array (welche üblicherweise durch die Funktionen **MultiLevelWaveletDecomposition2D** oder **MultiLevelMultipleWaveletDecomposition2D** erstellt werden). Es wird das maximale Analyselevel anhand der Filterhistorie ermittelt. Zudem werden aus dem kompletten Hochpass-Cell-Array die passenden Hochpassinformationen zu dem ausgewählten Tiefpassergebnis herausgesucht. Level für Level wird das Ausgangsbild nun aus den entsprechenden Tiefpass- und Hochpassinformationen und der passenden Rekonstruktionsfilterbank rekonstruiert. Dabei werden die bei der Analyse berechneten Verschiebungsvektoren, welche im Tiefpass-Cell-Array gespeichert sind, berücksichtigt. Die Funktion liefert die verwendeten Hochpassinformationen und alle Tiefpassteilergebnisse sowie das schließlich rekonstruierte Ursprungsbild zurück.
Anmerkung: Es werden zwar die Tiefpassergebnisse eines jeden Analyselevels und auch das Originalbild an die Rekonstruktionsfunktion übergeben, aber nur die Bildinformationen des übergebenen Tiefpass-Cell-Array-Elements werden für die Rekonstruktion tatsächlich verwendet.

Abbildung 2.14: Das Testbild 'Lena' wurde einer Analyse mit den zwei Filtern *OurInterpolM0_1* und *OurInterpolM1_1* unterzogen. Nun wird der Funktion **MultiLevelWaveletReconstruction2D** ein Element des Tiefpass-Cell-Arrays, das gesamte Tiefpass-Cell-Array und das gesamte Hochpass-Cell-Array übergeben. Die Abbildung veranschaulicht, wie das Ausgangsbild schrittweise wiederhergestellt wird und wie die Rückgabewerte aussehen. Bei jedem Filterergebnis ist zudem noch dessen Filterhistorie und der zugehörige Verschiebungsvektor gespeichert, welche aber der Übersicht halber nicht in der Abbildung enthalten sind.

2.1.15 MultiLevelWaveletReconstruction2D_1Level.m

```
function [lR, h1R, h2R] =
   MultiLevelWaveletReconstruction2D_1Level(l, lsV, h1, h1sV, h2, h2sV, F)
```

Die Funktion **MultiLevelWaveletReconstruction2D_1Level** erhält ein Tiefpassergebnis l und die zwei zugehörigen Hochpassergebnisse $h1$ und $h2$ sowie den Namen der Filterbank F der für die Analyse verwendet wurde und für die Synthese benützt werden soll. Zudem werden die während der Dekomposition berechneten Verschiebungsvektoren lsV, $h1sV$ und $h2sV$ für die einzelnen Zerlegungen übergeben.
Die Tiefpass- und Hochpassergebnisse werden jeweils einem Upsampling unterzogen und danach mit den Rekonstruktionsmatrizen der angegebenen Filterbank gefaltet. Anschließend werden auf die gefalteten Ergebnisse die Verschiebungsvektoren angewendet. Die Resultate werden schließlich zurückgegeben. Diese können dann direkt addiert werden, wodurch man das Ursprungsbild erhält.
Abbildung 2.15 zeigt auf, wie aus den Filterergebnissen eines Analyseschrittes das Originalbild wiederhergestellt wird.

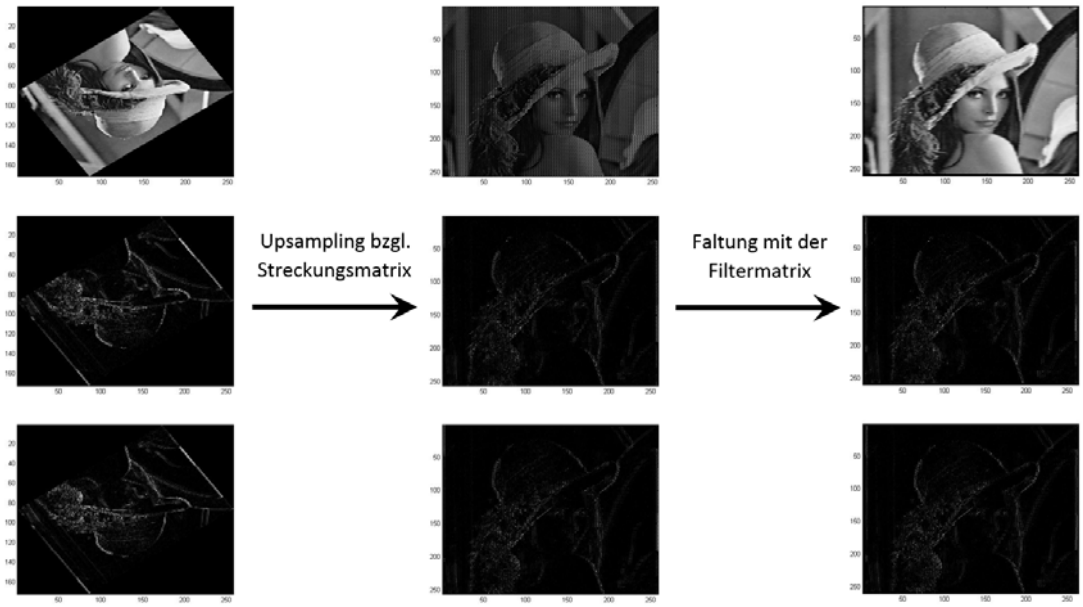


Abbildung 2.15: Die Abbildung verdeutlicht die Funktionsweise der Funktion **MultiLevelWaveletReconstruction2D_1Level**. Im ersten Schritt werden die einzelnen Filterergebnisse einem Upsampling unterzogen, im zweiten Schritt mit den entsprechenden Filtermatrizen der Filterbank gefaltet. Addiert man nun die Ergebnisse, erhält man das ursprüngliche Bild zurück.

2.1.16 FilterDatabase.m

```
function [M, L, H1, H2, sV] = FilterDatabase(N, P)
```

In der Funktion **FilterDatabase** sind die Koeffizienten ausgewählter Filterbänke gespeichert. Per Eingabe des Namens N einer Filterbank und des gewünschten Prozesses P (Analyse = *'decomposition'* oder Synthese = *'reconstruction'*) können die zugehörigen Koeffizienten von Hochpass- und Tiefpassfiltern abgerufen und zur weiteren Berechnung verwendet werden. Zudem wird zu jeder Filterbank die entsprechende Dilatationsmatrix fürs Sampling gespeichert und beim Aufruf übergeben. Die durch die Faltung hervorgerufenen Verschiebungsvektoren sV werden ebenfalls in einer Matrix übergegeben. Diese entsprechen dem Zweifachen der Repräsentanten der Streckungsmatrix, da sowohl bei der Analyse als auch bei der Synthese durch die Faltung eine Verschiebung um den entsprechenden Repräsentanten stattfindet. Diese beiden Verschiebungen müssen bei der Rekonstruktion wieder rückgängig gemacht werden. Genaueres zu den Repräsentanten der Streckungsmatrix und deren Berechnung findet sich im Abschnitt 2.4.2.
Abbildung 2.16 zeigt beispielhaft die gespeicherten Daten zu einer Filterbank.

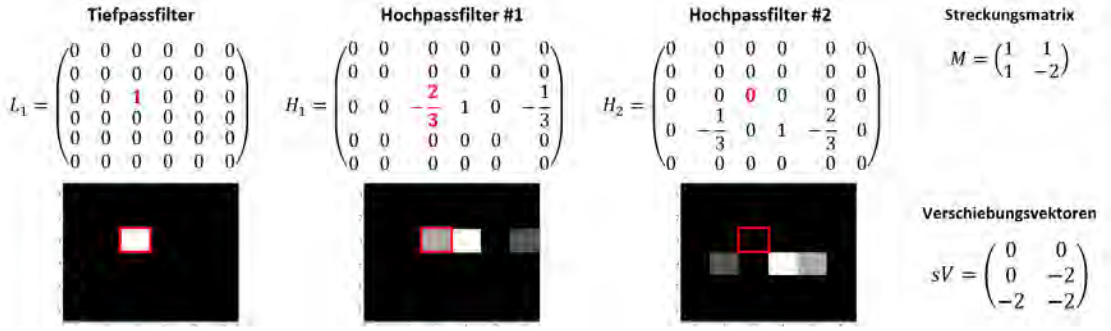


Abbildung 2.16: Diese Abbildung zeigt die Koeffizientenmatrizen der Filterbank *'Our Interpol M0_1'* und deren grafische Darstellung. Die Filtermatrix wird bei der Faltung über das zu filternde Bild geschoben. Bei diesem Vorgang werden Bildpunkte unter hellen Kästchen stark beeinflusst, Punkte unter dunklen Kästchen hingegen schwach.

2.2 Funktionsdefinitionen der C-Funktionen

2.2.1 MexFunctions.h

Struktur zur Speicherung einer Matrix

`struct mat`

Die Struktur **mat** speichert den Zeiger auf das erste Element einer *double*-Matrix und zudem deren Anzahl an Zeilen und Spalten. Dadurch kann auf jedes Element der in *Column-Major-Order* (siehe Abschnitt 3.2.1) gespeicherten Matrix zugegriffen werden.

Struktur zur Speicherung einer Matrix inklusive Verschiebungsvektor

`struct matWithShiftVector`

Die Struktur **matWithShiftVector** speichert den Zeiger auf das erste Element einer *double*-Matrix und deren Anzahl an Zeilen und Spalten. Zusätzlich kann der X- und Y-Versatz der Matrix in zwei Integerwerten der Struktur gespeichert werden.

Struktur zur Speicherung eines Verschiebungsvektors

`struct shiftVector`

Die Struktur **shiftVector** speichert die X- und Y-Koordinaten eines Verschiebungsvektors als Integerwerte.

Struktur zur Speicherung eines Filterergebnisses

```
struct filterResult
```

Die Struktur **filterResult** kann das Ergebnis eines 1-Level Filtervorgangs speichern. Dazu gehört das Tiefpassergebnis und die beiden Hochpassergebnisse. Intern wird jeweils der Zeiger auf das erste Element der *double*-Matrix und deren Anzahl an Zeilen und Spalten gespeichert. Für jedes Tiefpassergebnis und Hochpassergebnis wird zusätzlich der berechnete Verschiebungsvektor gespeichert.

matrixTranspose

```
double *matrixTranspose(double *M, size_t rows, size_t cols)
```

Die Funktion **matrixTranspose** erhält einen Zeiger $*M$ auf das erste Element einer Matrix, welche mit *double*-Werten belegt ist. Zusätzlich zum Zeiger wird die Anzahl der Reihen *rows* und Spalten *cols* der Matrix übergeben, sodass auf die in *Column-Major-Order* (siehe Abschnitt 3.2.1) gespeicherten Matrixelemente korrekt zugegriffen werden kann. Berechnet wird die Transponierte der übergebenen Matrix und als Rückgabewert erhält man wiederum einen Zeiger auf das erste *double*-Element der Matrix. Zeilen und Spalten der neuen Matrix müssen nicht extra zurückgegeben werden, da diese bei der Matrixtransposition nur vertauscht werden.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
transposedM = M';
```

matrixMultiplication

```
double *matrixMultiplication(double *A, double *B, size_t m,
   size_t p, size_t n)
```

Die Funktion **matrixMultiplication** erhält die Zeiger auf die beiden ersten Elemente $*A$ und $*B$ zweier *double*-Matrizen. Zudem werden die Matrixdimensionen übergeben, welche soweit übereinstimmen müssen, dass das Matrix-Produkt $matrixA \cdot matrixB$ gebildet werden kann.
Intern wird die Funktion *dgemm.c* aus der BLAS-Bibliothek aufgerufen, welche das Matrixprodukt schnell und effizient bildet.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
multipliedMatrices = A * B;
```

matrixDownsampling

```
struct mat *matrixDownsampling(double *M, int rows, int cols,
   int scaleX, int scaleY)
```

Die Funktion **matrixDownsampling** erhält einen Zeiger auf eine *double*-Matrix $*M$ und die Anzahl ihrer Zeilen *rows* und Spalten *cols*. Zudem werden die Skalierungsfaktoren *scaleX* und *scaleY* für das Downsampling in die jeweiligen Dimensionen übergeben. Man erhält eine Matrix zurück, welche im Sinne von Abschnitt 3.1.3 einem Downsampling unterzogen wurde. Zusätzlich zur Matrix werden auch deren Anzahl von Zeilen und Spalten in der Struktur *mat* übergeben.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
downsampledMatrix = Downsampling2D_1Dimension(M, 'X', scaleX);
downsampledMatrix = Downsampling2D_1Dimension(downsampledMatrix, 'Y', scaleY);
```

deleteZeroCols

```
struct mat *deleteZeroCols(double *M, int rows, int cols)
```

Die Funktion **deleteZeroCols** erhält den Zeiger auf eine *double*-Matrix M und deren Anzahl an Zeilen *rows* und Spalten *cols*. Die Matrix M wird auf Nullspalten zu Beginn und am Ende geprüft. Diese werden gegebenenfalls gelöscht. Die neu berechnete Matrix und deren Anzahl an Zeilen und Spalten wird in der Struktur *mat* zurückgegeben.
Die Funktion selbst ist mit keiner MatLab-Eingabe vergleichbar, da hier nur die Funktion **DeleteLastAndFirstZeroRowsAndColumns** existiert, welche zusätzlich die Nullzeilen zu Beginn und am Ende der Matrix löscht.

deleteZeroColumnsAndRows

```
struct mat *deleteZeroColumnsAndRows(double *M, int rows, int cols)
```

Die Funktion **deleteZeroColumnsAndRows** erhält den Zeiger auf eine *double*-Matrix $*M$ und deren Anzahl an Zeilen *rows* und Spalten *cols*. Mit Hilfe der Funktion **deleteZeroCols** werden zuerst die Nullspalten zu Beginn und am Ende der Matrix gelöscht. Danach wird die Matrix mittels **matrixTranspose** transponiert und es werden erneut die Nullspalten zu Beginn und am Ende gelöscht. Diese entsprechen den Nullzeilen der ursprünglichen Matrix, da sie zwischendurch transponiert wurde. Abschließend wird die resultierende Matrix rücktransponiert und mit ihrer Anzahl an Zeilen und Spalten in eine Struktur *mat* gepackt, welche zurückgegeben wird.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe (siehe Abschnitt 2.1.1):

```
slimMatrix = DeleteLastAndFirstZeroRowsAndColumns(M);
```

transformMatrix

```
struct mat *transformMatrix(double *M, int rows, int cols, double *T)
```

Die Funktion **transformMatrix** führt eine Transformation der Matrix $*M$ unter Verwendung der Transformationsmatrix T durch.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe (siehe Abschnitt 2.1.2):

```
transformedMatrix = TransformMatrix(M, T);
```

getFirstElementPositionOfMatrix

```
struct shiftVector getFirstElementPositionOfMatrix(double *M, int rows,
   int cols)
```

Die Funktion **getFirstElementPositionOfMatrix** ermittelt die Position des ersten Elements einer Matrix M, welches nicht Null ist.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe (siehe Abschnitt 2.1.4):

```
shiftVector = CalculateShiftVectorOfConvolution(M);
```

getPositionOfUniqueElementInMatrix

```
struct shiftVector getPositionOfUniqueElementInMatrix(double *M,
   int rows, int cols, int uniqueElement)
```

Die Funktion **getPositionOfUniqueElementInMatrix** gibt die Position eines eindeutigen Wertes in der Matrix M zurück.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
[rowVector, colVector, valueVector] = find(M);
orderedM = [rowVector, colVector, valueVector];
[~, row] = min(abs(orderedM(:, 3) - uniqueElement));
positionUniqueElement = [orderedM(row, 1), orderedM(row, 2)];
```

getMinimumValueOfMatrix

```
int getMinimumValueOfMatrix(double *M, int rows, int cols)
```

Die Funktion **getMinimumValueOfMatrix** liefert den kleinsten Wert einer üblicherweise positivwertigen Positionsmatrix zurück, wobei Nulleinträge nicht beachtet werden und ein Wert größer oder gleich 1 zurückgegeben wird.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
[~, ~, MValues] = find(M);
minimum = min(MValues(:,1));
```

calculateMatrixShiftVector

```
struct shiftVector calculateMatrixShiftVector(double *lM,
   int rows, int cols, int newMinimumValue)
```

Die Funktion **calculateMatrixShiftVector** erhält eine Positionsmatrix lM und einen neuen Minimalwert, welcher dem neuen $(0, 0)$-Element entspricht. Sie berechnet den Vektor zwischen dem alten $(0, 0)$-Element, dem der Wert 1 zugeordnet ist und dem neuen $(0, 0)$-Element.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
[rowVector, colVector, valueVector] = find(lM);
orderedlM = [rowVector, colVector, valueVector];
[~, row] = min(abs(orderedlM(:, 3) - 1));
positionOldZeroElement = [orderedlM(row, 1), orderedlM(row, 2)];
[~, row] = min(abs(orderedlM(:, 3) - newMinimumValue));
positionNewZeroElement = [orderedlM(row, 1), orderedlM(row, 2)];
shiftVector = positionNewZeroElement - positionOldZeroElement;
```

createLocatorImageFromMatrix

```
struct mat *createLocatorImageFromMatrix(double *M, int rows, int cols)
```

Die Funktion **createLocatorImageFromMatrix** erstellt eine Positionsmatrix derselben Gestalt wie die Matrix M, welche nur eindeutige positive aufsteigende Integerwerte enthält.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe:

```
locatorM = zeros(rows, cols);
locatorCounter = 1;
for(col = 1 : cols)
   for(row = 1 : rows)
      if(M(row, col) ~= 0)
         locatorM(row, col) = locatorCounter;
         locatorCounter = locatorCounter + 1;
      end
   end
end
```

WaveletDecomposition1Level

```
struct filterResult *WaveletDecomposition1Level(mxArray *image,
   const mxArray *lowpassFilter, const mxArray *highpassFilter1,
   const mxArray *highpassFilter2, int scaleFactorX,
   int scaleFactorY, double *smithFrontInv, double *smithBackInv)
```

Die Funktion **WaveletDecomposition1Level** erhält folgende Parameter:

- $mxArray$ *$image$ beinhaltet das zu filternden Bildes.
- $mxArray$ *$lowpassFilter$ enthält die Matrix des Tiefpassfilters, mit welchem das Bild gefiltert werden soll.
- $mxArray$ *$highpass1Filter$ enthält die Matrix des ersten Hochpassfilters, mit welchem das Bild gefiltert werden soll.
- $mxArray$ *$highpass2Filter$ enthält die Matrix des zweiten Hochpassfilters, mit welchem das Bild gefiltert werden soll.

- *intscaleFactorX* gibt den Faktor an, um welchen die Filterresultate in X-Richtung einem Downsampling unterzogen werden sollen.
- *intscaleFactorY* gibt den Faktor an, um welchen die Filterresultate in Y-Richtung einem Downsampling unterzogen werden sollen.
- *double *smithFrontInv* beinhaltet die Inverse der linken Matrix der Smith-Faktorisierung der Dilatationsmatrix des verwendeten Filters.
- *double *smithBackInv* beinhaltet die Inverse der rechten Matrix der Smith-Faktorisierung der Dilatationsmatrix des verwendeten Filters.

Die übergebenen Parameter werden bereits vor dem Aufruf der C-Funktionen noch in MatLab mit Hilfe der Funktion **SmithFactorization** berechnet.
Die Funktion **WaveletDecomposition1Level** berechnet nun die Filterergebnisse dieses 1-Level-Filterdurchganges und gibt jene in der entsprechenden Struktur *filterResult* zurück.
Anmerkung: Zur Berechnung der Faltungen wird aus der C-Funktion heraus über die MEX-Schnittstelle der zugehörige MatLab-Befehl *conv*2 aufgerufen.

Die Funktion ist vergleichbar mit folgender MatLab-Eingabe, wobei mit Hilfe des Parameters *filtername* die Filtermatrizen aus der Datei **FilterDatabase.m** abgerufen werden (siehe Abschnitt 2.1.13):

```
MultiLevelWaveletDecomposition2D_1Level(image, filtername)
```

2.2.2 WaveletDecompositionMex.c

```
[LL, HH] = WaveletDecompositionMex(image, filterName, lowpassFilter,
   highpass1Filter, highpass2Filter, decompositionLevel, scaleFactorX,
   scaleFactorY, smithFrontInv, smithBackInv);
```

Die MEX-Funktion **WaveletDecompositionMex** erhält folgende Parameter:

- *image* beinhaltet die Matrix des zu filternden Bildes.
- *filterName* gibt den Namen der verwendeten Filterbank an.
- *lowpassFilter* enthält die Matrix des Tiefpassfilters, mit welchem das Bild gefiltert werden soll.
- *highpass*1*Filter* enthält die Matrix des ersten Hochpassfilters, mit welchem das Bild gefiltert werden soll.
- *highpass*2*Filter* enthält die Matrix des zweiten Hochpassfilters, mit welchem das Bild gefiltert werden soll.
- *decompositionLevel* gibt die maximale Filtertiefe an.

- $scaleFactorX$ gibt den Faktor an, um welchen die Filterresultate in X-Richtung einem Downsampling unterzogen werden sollen.
- $scaleFactorY$ gibt den Faktor an, um welchen die Filterresultate in Y-Richtung einem Downsampling unterzogen werden sollen.
- $smithFrontInv$ beinhaltet die Inverse der linken Matrix der Smith-Faktorisierung der Dilatationsmatrix des verwendeten Filters.
- $smithBackInv$ beinhaltet die Inverse der rechten Matrix der Smith-Faktorisierung der Dilatationsmatrix des verwendeten Filters.

Die Funktion liefert ein Cell-Array mit den berechneten Tiefpass- und Hochpassresultaten zurück. Bei der Berechnung werden bis auf die Faltungsoperation im zweidimensionalen $conv2$ ausschließlich C-Funktionen verwendet.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe (siehe Abschnitt 2.1.12):

```
[LL, HH] = MultiLevelWaveletDecomposition2D(image, decompositionLevel,
   filterName);
```

2.2.3 MultipleWaveletDecompositionMex.c

```
[LL, HH] = MultipleWaveletDecompositionMex(image, filterAName, lowpassFilterA,
   highpass1FilterA, highpass2FilterA, filterBName, lowpassFilterB,
   highpass1FilterB, highpass2FilterB, decompositionLevel, scaleFactorAX,
   scaleFactorAY, smithFrontAInv, smithBackAInv, scaleFactorBX, scaleFactorBY,
   smithFrontBInv, smithBackBInv);
```

Die MEX-Funktion **MultipleWaveletDecompositionMex** erhält folgende Parameter:

- $image$ beinhaltet die Matrix des zu filternde Bild.
- $filterAName$ gibt den Namen der ersten verwendeten Filterbank an.
- $lowpassFilterA$ enthält die Matrix des Tiefpassfilters der Filterbank $filterAName$, mit welchem das Bild gefiltert werden soll.
- $highpass1FilterA$ enthält die Matrix des ersten Hochpassfilters der Filterbank $filterAName$, mit welchem das Bild gefiltert werden soll.
- $highpass2FilterA$ enthält die Matrix des zweiten Hochpassfilters der Filterbank $filterAName$, mit welchem das Bild gefiltert werden soll.
- $filterBName$ gibt den Namen der zweiten verwendeten Filterbank an.
- $lowpassFilterB$ enthält die Matrix des Tiefpassfilters der Filterbank $filterBName$, mit welchem das Bild gefiltert werden soll.

- $highpass1FilterB$ enthält die Matrix des ersten Hochpassfilters der Filterbank $filterBName$, mit welchem das Bild gefiltert werden soll.
- $highpass2FilterB$ enthält die Matrix des zweiten Hochpassfilters der Filterbank $filterBName$, mit welchem das Bild gefiltert werden soll.
- $decompositionLevel$ gibt die maximale Filtertiefe an.
- $scaleFactorAX$ gibt den Faktor an, um welchen die Filterresultate der Filterbank $filterAName$ in X-Richtung einem Downsampling unterzogen werden sollen.
- $scaleFactorAY$ gibt den Faktor an, um welchen die Filterresultate der Filterbank $filterAName$ in Y-Richtung einem Downsampling unterzogen werden sollen.
- $smithFrontAInv$ beinhaltet die Inverse der linken Matrix der Smith-Faktorisierung der Dilatationsmatrix der Filterbank $filterAName$.
- $smithBackAInv$ beinhaltet die Inverse der rechten Matrix der Smith-Faktorisierung der Dilatationsmatrix der Filterbank $filterAName$.
- $scaleFactorBX$ gibt den Faktor an, um welchen die Filterresultate der Filterbank $filterBName$ in X-Richtung einem Downsampling unterzogen werden sollen.
- $scaleFactorBY$ gibt den Faktor an, um welchen die Filterresultate der Filterbank $filterBName$ in Y-Richtung einem Downsampling unterzogen werden sollen.
- $smithFrontBInv$ beinhaltet die Inverse der linken Matrix der Smith-Faktorisierung der Dilatationsmatrix der Filterbank $filterBName$.
- $smithBackBInv$ beinhaltet die Inverse der rechten Matrix der Smith-Faktorisierung der Dilatationsmatrix der Filterbank $filterBName$.

Die Funktion liefert ein Cell-Array mit den berechneten Tiefpass- und Hochpassresultaten zurück. Bei der Berechnung werden bis auf die Faltungsoperation im zweidimensionalen $conv2$ ausschließlich C-Funktionen verwendet.
Die Funktion ist vergleichbar mit folgender MatLab-Eingabe (siehe Abschnitt 2.1.11):

```
[LL, HH] = MultiLevelMultipleWaveletDecomposition2D(image,
   decompositionLevel, filterAName, filterBName);
```

2.3 Behandlung des Verschiebungsvektors

Sowohl die Faltung eines Bildes mit einem Filter als auch das Downsampling des Bildes können zu Verschiebungen der Bildinformationen führen. Im Folgenden wird erläutert, wie in der Implementierung auf diese Verschiebung reagiert wird. Die Ausführung wird mit MatLab-Code-Beispielen untermauert, in der C-Implementierung wird analog vorgegangen.

Sei dazu $M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ und $filter$ = 'OurInterpolM0_1'.

- Filterkoeffizienten für den Analyseteil aus der Filterdatenbank abrufen

```
[dilationMatrix, lowpassFilter, highpassFilter1, highpassFilter2] = ...
   FilterDatabase(filter, 'decomposition');
```

- Faltung der Matrix M mit den Filtermatrizen durchführen

```
l = conv2(double(M), double(lowpassFilter));
h1 = conv2(double(M), double(highpassFilter1));
h2 = conv2(double(M), double(highpassFilter2));
```

Achtung: Bei der Faltung darf unter keinen Umständen das Schlüsselwort '*same*' als dritter Parameter der *conv*2-Funktion übergeben werden. Dadurch wird nur der zentrale Teil des Faltungsprodukts zurückgegeben. So werden berechnete Werte an den Rändern vernachlässigt, was zu einem Informationsverlust führt. Die Verwendung von '*same*' als dritter Parameter macht eine perfekte Rekonstruktion unmöglich.

Man erhält für die Tiefpass- und Hochpassergebnisse:

$$- \; l = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 & 0 & 0 \\ 0 & 0 & 5 & 6 & 7 & 8 & 0 & 0 \\ 0 & 0 & 9 & 10 & 11 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \; h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0,6667 & -0,3333 & 0 & 0 & 3,3333 & -1 & -1,3333 \\ 0 & 0 & -3,3333 & 1 & 1,3333 & 0 & 6 & -2,3333 & -2,6667 \\ 0 & 0 & -6 & 2,3333 & 2,6667 & 0 & 8,6667 & -3,6667 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \; h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0,3333 & -0,6667 & 0 & 0 & 1,6667 & 2 & -2,6667 & 0 \\ 0 & -1,6667 & -2 & 2,6667 & 0 & 3 & 3,3333 & -5,3333 & 0 \\ 0 & -3 & -3,3333 & 5,3333 & 0 & 4,3333 & 4,6667 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Berechnung der Verschiebung der Bildmatrizen

```
lsV = CalculateShiftVectorOfConvolution(l);
h1sV = CalculateShiftVectorOfConvolution(h1);
h2sV = CalculateShiftVectorOfConvolution(h2);
```

  Man erhält für die Verschiebungsvektoren folgende Werte, welche man auch ganz einfach von den Filterergebnissen ablesen kann:

  - $lsV = \begin{pmatrix} 2 & 2 \end{pmatrix}$
  - $h1sV = \begin{pmatrix} 2 & 2 \end{pmatrix}$
  - $h2sV = \begin{pmatrix} 3 & 1 \end{pmatrix}$

- Downsampling und Transformation Die gefalteten Matrizen werden einem Downsampling unterzogen. Während dieses Prozesses müssen die Verschiebungsvektoren ständig aktualisiert werden. Nach dem Downsampling wird die gesampelte Matrix und der zusätzliche Verschiebungsvektor zurückgegeben.

```
[l, lsV'] = Downsampling2D_DilationMatrix_...
   ReturningAdditionalShiftVector(l, dilationMatrix);
[h1, h1sV'] = Downsampling2D_DilationMatrix_...
   ReturningAdditionalShiftVector(h1, dilationMatrix);
[h2, h2sV'] = Downsampling2D_DilationMatrix_...
   ReturningAdditionalShiftVector(h2, dilationMatrix);
```

  Dieser Vorgang wird beispielhaft anhand des Downsamplings des Hochpasses $h1$ erläutert:

  - Erstellung einer Positionsmatrix
    Die Positionsmatrix hat dieselbe Gestalt wie die Ausgangsmatrix, allerdings enthält sie aufsteigend nummerierte eindeutige Einträge. Die Nummerierung erfolgt in Column-Major-Order. Die Positionsmatrix wird zusätzlich in der Variable $h1LocOriginal$ gespeichert, da von der ursprünglichen Gestalt der Verschiebungsvektor abgelesen werden kann.

$$h1Loc = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 0 & 0 & 9 & 12 & 15 \\ 0 & 0 & 2 & 5 & 7 & 0 & 10 & 13 & 16 \\ 0 & 0 & 3 & 6 & 8 & 0 & 11 & 14 & 17 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = h1LocOriginal$$

  - Transformation von Matrix und Positionsmatrix
    Sowohl die eigentliche Matrix $h1$ als auch deren Positionsmatrix $h1Loc$ werden vor dem Downsampling derselben Transformation unterzogen.
    Man erhält folgende Matrizen:

$$h1 = \begin{pmatrix} -4 & 0 & 0 \\ -3,6667 & 0 & 0 \\ 8,6667 & 0 & 0 \\ 0 & 0 & 0 \\ 2,6667 & -2,6667 & 0 \\ 2,3333 & -2,3333 & 0 \\ -6 & 6 & 0 \\ 0 & 0 & 0 \\ 0 & 1,3333 & -1,3333 \\ 0 & 1 & -1 \\ 0 & -3,3333 & 3,3333 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -0,3333 \\ 0 & 0 & -0,6667 \end{pmatrix} \text{ und } h1Loc = \begin{pmatrix} 17 & 0 & 0 \\ 14 & 0 & 0 \\ 11 & 0 & 0 \\ 0 & 0 & 0 \\ 8 & 16 & 0 \\ 6 & 13 & 0 \\ 3 & 10 & 0 \\ 0 & 0 & 0 \\ 0 & 7 & 15 \\ 0 & 5 & 12 \\ 0 & 2 & 9 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

- Downsampling von Matrix und Positionsmatrix
  Beide Matrizen werden nun einem Downsampling unterzogen. Die Samplingfaktoren stammen von der Dilatationsmatrix. Diese schreibt vor, dass die Spaltenzahl unverändert bleibt, aber nur jede dritte Zeile beibehalten wird. Als Ergebnis nach dem Downsampling ergibt sich:

  $$h1 = \begin{pmatrix} -4 & 0 & 0 \\ 0 & 0 & 0 \\ -6 & 6 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \text{ und } h1Loc = \begin{pmatrix} 17 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 10 & 0 \\ 0 & 5 & 12 \\ 0 & 0 & 0 \end{pmatrix}$$

  Anmerkung: Bereits hier wird ersichtlich, dass unser ursprüngliches $(0, 0)$-Element der Matrix dem Downsampling zum Verhängnis wurde. Um die Matrix also für die Rekonstruktion an die richtige Stelle zu schieben, muss der bereits gespeicherte Verschiebungsvektor abgeändert werden.
- Rücktransformation von Matrix und Positionsmatrix
  Sowohl die eigentliche Matrix $h1$ als auch deren Positionsmatrix $h1Loc$ werden nach dem Downsampling derselben Rücktransformation unterzogen.
  Man erhält folgende Matrizen:

  $$h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 6 & -1 \\ -6 & 1 & 0 & 0 & 0 \end{pmatrix} \text{ und } h1Loc = \begin{pmatrix} 0 & 0 & 0 & 0 & 17 \\ 0 & 0 & 0 & 10 & 12 \\ 3 & 5 & 0 & 0 & 0 \end{pmatrix}$$

- Berechnung des Verschiebungsvektors

```
[~, ~, h1LocValues] = find(h1Loc);
newZeroValue = min(h1LocOriginalValues(:,1));
```

```
[rowVector, colVector, valueVector] = find(h1LocOriginal);
orderedH1LocOriginal = [rowVector, colVector, valueVector];

[~, row] = min(abs(orderedH1LocOriginal(:, 3) - 1));
positionOldZero = [orderedH1LocOriginal(row, 1),orderedH1LocOriginal(row, 2)];

[~, row] = min(abs(orderedH1LocOriginal(:, 3) - newZeroElement));
positionNewZero = [orderedH1LocOriginal(row, 1),orderedH1LocOriginal(row, 2)];

additionalShift = positionNewZeroElement - positionOldZeroElement;
h1sV = h1sV + additionalShift;
```

Das neue $(0,0)$-Element befindet sich an der Position in der Matrix $h1Loc$, an welcher das kleinste Element steht. Im Beispielfall gilt:
$newZeroValue = 3$
Das alte $(0,0)$-Element der Matrix, welches mit der Position des Wertes 1 in der Positionsmatrix identifiziert wird, besitzt natürlich die Position
$positionOldZero = \begin{pmatrix}3 & 3\end{pmatrix}$ (siehe $h1LocOriginal$).
Das neue $(0,0)$-Element befindet sich an der Stelle, wo der Wert $newZeroValue = 3$ in der Positionsmatrix steht.

Es gilt also: $positionNewZero = \begin{pmatrix}5 & 3\end{pmatrix}$.

Dadurch lässt sich nun der aktualisierte Verschiebungsvektor berechnen:

$additionalShift = \begin{pmatrix}5 & 3\end{pmatrix} - \begin{pmatrix}3 & 3\end{pmatrix} = \begin{pmatrix}2 & 0\end{pmatrix}$
$h1sV = \begin{pmatrix}2 & 2\end{pmatrix} + \begin{pmatrix}2 & 0\end{pmatrix} = \begin{pmatrix}4 & 2\end{pmatrix}$

Analog lassen sich die Werte für den Tiefpassfilter und den zweiten Hochpassfilter berechnen.
Man erhält folgende Tief- und Hochpassfilterergebnisse samt aktualisiertem Verschiebungsvektor nach dem Downsampling:

- $l = \begin{pmatrix}0 & 0 & 12\\ 9 & 6 & 3\end{pmatrix}$ mit $lsV = \begin{pmatrix}4 & 2\end{pmatrix}$
- $h1 = \begin{pmatrix}0 & 0 & 0 & 0 & -4\\ 0 & 0 & 0 & 6 & -1\\ -6 & 1 & 0 & 0 & 0\end{pmatrix}$ mit $h1sV = \begin{pmatrix}4 & 2\end{pmatrix}$
- $h2 = \begin{pmatrix}0 & 0 & 0 & 0 & -8\\ 0 & 0 & 0 & 3 & 2\\ -3 & -2 & 0 & 0 & 0\end{pmatrix}$ mit $h2sV = \begin{pmatrix}5 & 1\end{pmatrix}$

Ein kompletter Analyseschritt ist nun vollendet. Die gewonnenen Daten reichen aus, um die Originalmatrix perfekt zu rekonstruieren, was im Folgenden geschieht.

- Filterkoeffizienten für den Analyseteil aus der Filterdatenbank abrufen

```
[dilationMatrix, lowpassFilter, highpassFilter1, highpassFilter2] = ...
    FilterDatabase(filter, 'reconstruction');
```

- Upsamling der Filterresultate

```
l = Upsampling2D_DilationMatrix(l, dilationMatrix);
h1 = Upsampling2D_DilationMatrix(h1, dilationMatrix);
h2 = Upsampling2D_DilationMatrix(h2, dilationMatrix);
```

  Vor der Rekonstruktion werden die Tiefpass- und Hochpassinformationen einem Upsampling unterzogen, wobei man folgende Matrizen erhält:

  - $l = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \\ 9 & 0 & 0 & 12 \end{pmatrix}$
  - $h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 6 & 0 & 0 \\ -6 & 0 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}$
  - $h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 & 3 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 0 & -8 \end{pmatrix}$

- Anwendung des Verschiebungsvektors auf die Filterresultate

```
l = ApplyShiftVectorToMatrix(l, lsV);
h1 = ApplyShiftVectorToMatrix(h1, h1sV);
h2 = ApplyShiftVectorToMatrix(h2, h2sV);
```

  Die Daten wurden einem Upsampling unterzogen, können aber noch nicht direkt mit den Rekonstruktionsfiltern der Filterbank gefaltet werden. Denn zuerst müssen die Matrizen korrekt ausgerichtet werden. Das bedeutet, dass das $(0, 0)$-Element dieselbe Position einnehmen muss wie vor dem Downsampling.
  Um dies zu erreichen, muss lediglich der aktualisierte Verschiebungsvektor auf die Matrizen angewandt werden. Dadurch wird entweder die korrekte Anzahl an Nullzeilen und -spalten zu Beginn der Matrizen eingefügt oder die entsprechende Anzahl an Zeilen und Spalten zu Beginn der Matrizen gelöscht. Die Nullzeilen und -spalten am Ende der Matrizen sind nebensächlich, da die Matrizen am linken oberen Eck ausgerichtet werden.
  Nach der Verschiebung erhält man:

  - $l = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 12 \end{pmatrix}$

$$- \ h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & -6 & 0 & 0 & 0 & 0 & 0 & -4 \end{pmatrix}$$

$$- \ h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & -2 & 0 & 0 & 3 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 0 & -8 \end{pmatrix}$$

- Faltung der Tiefpass- und Hochpassinformationen mit den Rekonstruktionsfiltermatrizen:

```
l = conv2(double(l), double(lowpassFilter));
h1 = conv2(double(h1), double(highpassFilter1));
h2 = conv2(double(h2), double(highpassFilter2));
```

Achtung: Auch hier ist auf den Parameter '*same*' zu verzichten. Bei Nichtangabe des dritten Parameters der Funktion *conv*2 wird automatisch '$full$' verwendet, was die komplette Faltungsmatrix zurückgibt.
Als Ergebnis erhält man:

$$- \ l = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 4 & 6 & 4 & 2 & 0 & 0 \\ 0 & 0 & 3 & 6 & 9 & 10 & 11 & 12 & 8 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \ h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \ h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Berücksichtigung der Verschiebung durch die Filtermatrizen

```
l = RelocateZeroElement(l, shiftVectors(1,:));
h1 = RelocateZeroElement(h1, shiftVectors(2,:));
h2 = RelocateZeroElement(h2, shiftVectors(3,:));
```

Die Matrizen bzw. deren Inhalte werden durch die Faltung mit einer Filtermatrix in der Regel verschoben. Der Verschiebungsvektor ist abhängig vom Filter und für jede Filtermatrix in der Filterdatenbank hinterlegt. Im Detail entspricht der Verschiebungsvektor dem Zweifachen des Repräsentanten der Dilatationsmatrix, welcher den aktuellen Filter erzeugt (siehe dazu Abschnitt 2.4.2). Die Matrixeinträge müssen um das Zweifache des Vektors zurückverschoben werden, da sie ja sowohl bei der Analyse als auch bei der Synthese jeweils einmal um den Verschiebungsvektor verschoben werden.
Man erhält folgende rückverschobene Matrizen:

$$- \ l = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 4 & 6 & 4 & 2 & 0 & 0 \\ 0 & 0 & 3 & 6 & 9 & 10 & 11 & 12 & 8 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \ h1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$- \ h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Addition der Teilinformationen

  ```
  M = l + h1 + h2;
  ```

  Dass die einzelnen Matrizen vor der Addition noch auf dieselbe Größe gebracht werden müssen, indem man am Ende Nullzeilen und -spalten einfügt, liegt auf der Hand. MatLab kann Matrizen mit unterschiedlicher Größe nicht intuitiv durch Auffüllen mit Nullen addieren.
  Als rekonstruierte Matrix M erhält man:

$$l + h1 + h2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 & 7 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 10 & 11 & 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Entfernung von Nullzeilen und -spalten

  ```
  M = DeleteLastAndFirstZeroRowsAndColumns(round(M));
  ```

  Die Nullzeilen und -spalten zu Beginn und am Ende der rekonstruierten Matrix werden entfernt. Zugleich werden die Einträge der Matrix auf Ganzzahlen gerundet. Denn bei der Analyse und Synthese können kleinere Rundungsfehler auftreten. Da die Farbwerte eines Bildes hier aber in Ganzzahlen angegeben werden, werden dadurch am Schluss der Prozedur diese Ungenauigkeitsfehler entfernt.
  Und wie erwartet konnte die ursprüngliche Matrix M perfekt rekonstruiert werden:

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

2.4 Mathematische Hintergründe

2.4.1 Smith-Faktorisierung

Definition 2.4.1 Smith-Normalform [2]
Eine Matrix $A \in R^{m \times n}$ über einem Hauptidealring R mit Rang $r > 0$ ist in Smith-Normalform (SNF), wenn folgende Bedingungen erfüllt sind:

(i) A ist in Diagonalform

(ii) Diagonalelemente $a_{i,i} \neq 0$ für $1 \leq i \leq r$

(iii) Diagonalelemente $a_{i,i} = 0$ für $i > r$

(iv) Die Diagonalelemente stellen eine umgekehrte Teilerkette dar, d.h. $a_{i,i} | a_{i+1,i+1}$ für $1 \leq i \leq r-1$

Beispiel 2.4.2 Matrix in Smith-Normalform
Die Matrix $A \in \mathbb{Z}^{4 \times 5}$ mit Rang $r = 3$ ist in Smith-Normalform.

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & -12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Satz 2.4.3 Smith-Faktorisierung [2]
Für eine Matrix $C \in R^{m \times n}$ über einem Hauptidealring R existieren invertierbare quadratische Matrizen $U \in R^{m \times m}$ und $V \in R^{n \times n}$, sodass das Matrixprodukt $U \cdot C \cdot V$ in Smith-Normalform ist.

$$U \cdot C \cdot V = D := \begin{pmatrix} a_{i,i} & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & a_{r,r} & \ddots & & \vdots \\ \vdots & & \ddots & 0 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & \dots & 0 & 0 \end{pmatrix} \text{ bzw. } C = U^{-1} \cdot D \cdot V^{-1}$$

Definition 2.4.4 Benennung zur SNF [2]
Die Diagonalelemente $a_{1,1}, ..., a_{r,r}$ der Matrix A in SNF heißen Elementarteiler. Sie sind bis auf Multiplikation mit einer Einheit eindeutig definiert. Die Matrizen U, V heißen linke und rechte Transformationsmatrix zur Smith-Normalform.

Bemerkung 2.4.5 Berechnung einer Smith-Faktorisierung
Die Smith-Faktorisierung kann zum Beispiel mit Hilfe der Gauß-Elimination und Division mit Rest berechnet werden. In Abschnitt 3.1 wird der Einsatz und die Berechnung der Faktorisierung in MatLab beschrieben.

2.4.2 Quotientengruppen

Definiton 2.4.6 Restklassenring [3]
Für eine natürliche Zahl n fasst man ganze Zahlen mit gleichem Rest bei Division durch diese Zahl n zu Restklassen zusammen. Diese Restklassen bilden zusammen den Restklassenring $\mathbb{Z}/n\mathbb{Z}$ (sprich „Z modulo nZ“).

Beispiel 2.4.7 Restklassenring
Für den Restklassenring $\mathbb{Z}/7\mathbb{Z}$ kann als einfachstes Repräsentantensystem $\{0, 1, 2, 3, 4, 5, 6\}$ angegeben werden. Der Ring hat genau 7 Elemente, da bei der Division einer ganzen Zahl durch 7 genau 7 mögliche verschiedene Restwerte auftreten können.

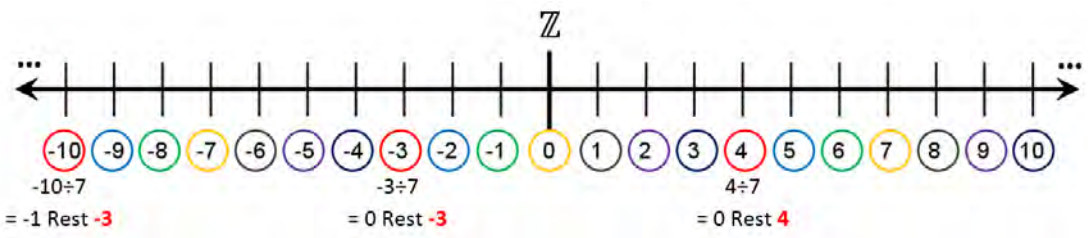


Abbildung 2.17: Farbig gekennzeichnete Restklassen des Restklassenrings $\mathbb{Z}/7\mathbb{Z}$. Elemente der gleichen Farbe gehören zu einer Restklasse. Im Ring $\mathbb{Z}/7\mathbb{Z}$ entspricht zum Beispiel der Rest -3 dem Rest 4.

Bemerkung 2.4.8 Quotientengruppen im Mehrdimensionalen [1]
Man kann auch Quotientengruppen von $\mathbb{Z}^s$ mit $s \in \mathbb{N}$, insbesondere auch von $\mathbb{Z}^2$, bilden. Dabei wird nicht mehr mit einer natürlichen Zahl modulo gerechnet, sondern mit einer ganzzahligen Matrix $\Xi \in \mathbb{Z}^{s\times s}$.

Satz 2.4.9 Quotientengruppen in $\mathbb{Z}^2$ [1]
Die Quotientengruppe $E_\Xi := \mathbb{Z}^2/\Xi\mathbb{Z}^2$ wird von den $|det(\Xi)|$ Elementen aus $\Xi \cdot [0, 1)^2 \cap \mathbb{Z}^2$ generiert.

Bemerkung 2.4.10
Im s-dimensionalen Fall besteht ein Repräsentantensystem einer Quotientengruppe $\mathbb{Z}^s/\Xi\mathbb{Z}^s$ aus $|det(\Xi)|$ Elementen, wobei Ξ eine $(s \times s)$-Matrix ist.
Dies gilt natürlich auch im eindimensionalen Fall, denn $\mathbb{Z}/n\mathbb{Z}$ mit $n \in \mathbb{N}$ wird genau von $|det(n)| = n$ Elementen repräsentiert.

Beispiel 2.4.11 Berechnung eines Repräsentantensystems einer Quotientengruppe [1]
Im Folgenden wird eine Methode zur Berechnung eines Repräsentantensystems für die Quotientengruppe $\mathbb{Z}^s/\Xi\mathbb{Z}^s$ angegeben und für $s = 1$ und $s = 2$ exemplarisch berechnet. Gegeben ist eine $s \times s$-Matrix $\Xi \neq 0$.

1. Wähle $N = \|\Xi\|_\infty = \max\limits_{i=1,\dots,s} \sum\limits_{k=1}^{s} |\xi_{i,k}|$, sodass $\Xi \cdot [0,1) \subset [-N,N]^s$ gilt.

2. Setze $\Omega = [-N,N]^s$ und $E_\Xi = \emptyset$ und wiederhole folgende Teilschritte solange $\Omega \neq \emptyset$ gilt.
   (Teilschritte werden mindestens einmal ausgeführt, da Ω von Beginn an wegen $\Xi \neq 0$ ungleich der leeren Menge ist.)
   a) Wähle einen beliebigen Vektor $\omega \in \Omega$
   b) Füge $\omega - \Xi \cdot \lfloor \Xi^{-1} \cdot \omega \rfloor$ als Repräsentant zum Repräsentantensystem E_Ξ hinzu.
   c) Entferne alle Repräsentanten $\Omega \cap (\omega + \Xi\mathbb{Z}^s)$ aus dieser Restklasse"von der Menge Ω

- **Eindimensionaler Fall:** $s = 1$
  Betrachte die (1×1)-Matrix $\Xi = 3$ und deren Inverse $\Xi^{-1} = \frac{1}{3}$.
  1. $N = \|3\|_\infty = 3$ und es gilt $\Xi \cdot [0,1) = 3 \cdot [0,1) = [0,3) \subset [-N,N]^1 = [-3,3]$
  2. $\Omega = [-3,3] = \{-3,-2,-1,0,1,2,3\}$ und setze $E_\Xi = \emptyset$
     - **1. Durchgang:** Wähle zufällig Vektor $\omega = -3$ und füge
       $\omega - \Xi \cdot \lfloor \Xi^{-1} \cdot \omega \rfloor = -3 - 3 \cdot \lfloor (3)^{-1} \cdot (-3) \rfloor = -3 - 3 \cdot \lfloor \frac{1}{3} \cdot (-3) \rfloor = -3 - 3 \cdot \lfloor -1 \rfloor = -3 - 3 \cdot (-1) = -3 + 3 = 0$
       als Repräsentant zu E_Ξ hinzu.
       Entferne $\Omega \cap (\omega + \Xi\mathbb{Z}^s) = \{-3,-2,-1,0,1,2,3\} \cap (-3 + 3\mathbb{Z})$
       $= \{-3,-2,-1,0,1,2,3\} \cap \{\dots,-9,-6,-3,0,3,6,9,\dots\} = \{-3,0,3\}$ von Ω und erhalte somit $\Omega = \{-2,-1,1,2\}$.
     - **2. Durchgang:** Wähle zufällig Vektor $\omega = 1$ und füge
       $1 - 3 \cdot \lfloor (3)^{-1} \cdot 1 \rfloor = 1 - 3 \cdot \lfloor \frac{1}{3} \cdot 1 \rfloor = 1 - 3 \cdot \lfloor \frac{1}{3} \rfloor = 1 - 3 \cdot 0 = 1 - 0 = 1$
       als Repräsentant zu E_Ξ hinzu.
       Entferne $\{-2,-1,1,2\} \cap (1 + 3\mathbb{Z}) = \{-2,-1,1,2\} \cap \{\dots,-5,-2,1,4,\dots\} = \{-2,1\}$ von Ω und erhalte somit $\Omega = \{-1,2\}$.
     - **3. Durchgang:** Wähle zufällig Vektor $\omega = -1$ und füge
       $-1 - 3 \cdot \lfloor \frac{1}{3} \cdot (-1) \rfloor = -1 - 3 \cdot \lfloor -\frac{1}{3} \rfloor = -1 - 3 \cdot (-1) = -1 + 3 = 2$
       als Repräsentant zu E_Ξ hinzu.
       Entferne $\{-1,2\} \cap (2 + 3\mathbb{Z}) = \{-1,2\} \cap \{\dots,-4,-1,2,5,\dots\} = \{-1,2\}$ von Ω und erhalte somit $\Omega = \emptyset$.
  3. Erhalte $E_\Xi = \{0,1,2\}$ wie erwartet als Repräsentantensystem für $\mathbb{Z}/3\mathbb{Z}$.

- **Zweidimensionaler Fall:** $s = 2$
  Betrachte die (2×2)-Matrix $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ und deren Inverse
  $\Xi^{-1} = \frac{1}{det(\Xi)} \cdot \begin{pmatrix} -2 & -1 \\ -1 & 1 \end{pmatrix} = -\frac{1}{3} \cdot \begin{pmatrix} -2 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} \end{pmatrix}$.

  1. $N = \left\| \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \right\|_\infty = \max\limits_{i=1,2} \sum\limits_{k=1}^{2} |\xi_{i,k}| = \max\{2, 3\} = 3$ und es gilt $\Xi \cdot [0,1)^2 = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot [0,1)^2 \subset [-3,3]^2$

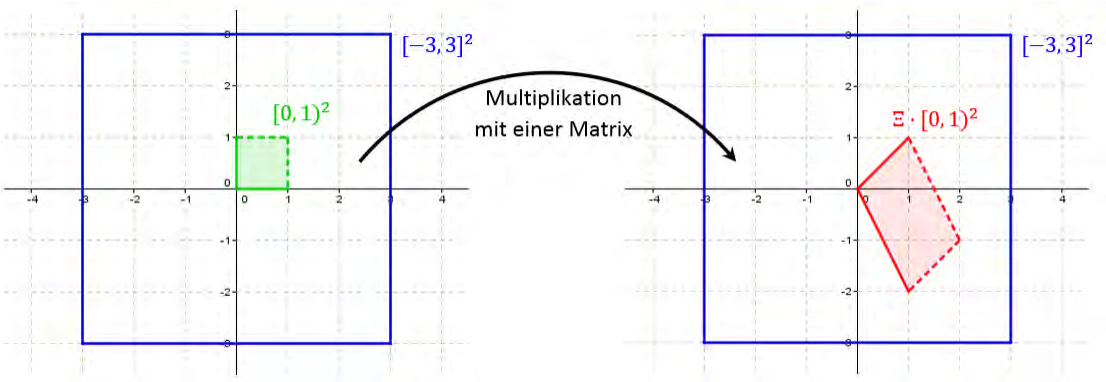


Abbildung 2.18: Links ist die Fläche $[0,1)^2$ eingebettet in $\Omega = [-3,3]^2$ dargestellt. Rechts sieht man, dass die Matrix Ξ die Fläche $[0,1)^2$ zu einem Parallelogramm verformt, welches jedoch immer noch Teilmenge von Ω ist.

  2. $\Omega = [-3,3]^2 = \left\{ \begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} -3 \\ 2 \end{pmatrix}, \begin{pmatrix} -3 \\ 1 \end{pmatrix}, \begin{pmatrix} -3 \\ 0 \end{pmatrix}, ..., \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}$
     und setze $E_\Xi = \emptyset$

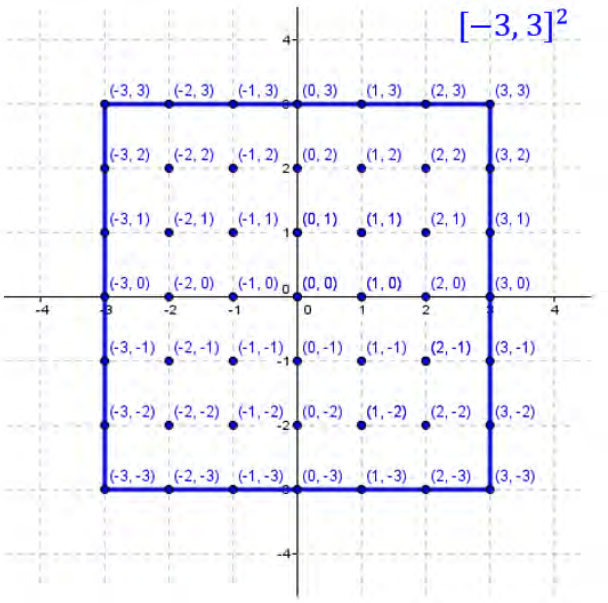


Abbildung 2.19: Die Abbildung zeigt alle Elemente von $\Omega = [-3,3]^2$.

– **1. Durchgang:** Wähle zufällig Vektor $\omega = \begin{pmatrix} -3 \\ -3 \end{pmatrix}$ und berechne

$$\omega - \Xi \cdot \lfloor \Xi^{-1} \cdot \omega \rfloor = \begin{pmatrix} -3 \\ -3 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \left\lfloor \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} -3 \\ -3 \end{pmatrix} \right\rfloor =$$
$$\begin{pmatrix} -3 \\ -3 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \left\lfloor -\tfrac{1}{3} \cdot \begin{pmatrix} -2 & -1 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} -3 \\ -3 \end{pmatrix} \right\rfloor =$$
$$\begin{pmatrix} -3 \\ -3 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \left\lfloor -\tfrac{1}{3} \cdot \begin{pmatrix} 9 \\ -3 \end{pmatrix} \right\rfloor = \begin{pmatrix} -3 \\ -3 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \left\lfloor \begin{pmatrix} -3 \\ 1 \end{pmatrix} \right\rfloor =$$
$$\begin{pmatrix} -3 \\ -3 \end{pmatrix} - \begin{pmatrix} -2 \\ -5 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

Füge nun $\begin{pmatrix} -1 \\ 2 \end{pmatrix}$ als Repräsentant seiner „Restklasse“hinzu und streiche alle Repräsentanten dieser Restklasse aus Ω:
Folgende Elemente müssen demnach gestrichen werden:

$$\Omega \cap (\omega + \Xi\mathbb{Z}^2) = \Omega \cap \left(\begin{pmatrix} -3 \\ -3 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \mathbb{Z}^2 \right) =$$
$$\Omega \cap \left(\begin{pmatrix} -3 \\ -3 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right) = \Omega \cap \begin{pmatrix} z_1 + z_2 - 3 \\ z_1 - 2z_2 - 3 \end{pmatrix} =$$
$$\Omega \cap \left\{ ..., \begin{pmatrix} -4 \\ 2 \end{pmatrix}, \begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} -2 \\ 4 \end{pmatrix}, ..., ..., \begin{pmatrix} -4 \\ -1 \end{pmatrix}, \begin{pmatrix} -3 \\ 0 \end{pmatrix}, \begin{pmatrix} -2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 4 \end{pmatrix}, ... \right\}$$

Abbildung 2.20 visualisiert die einzelnen Durchläufe und zeigt auf, bei welchem Durchlauf welche Elemente aus Ω gestrichen werden.

– **2. und 3. Durchgang:** Funktionieren analog zum 1. Durchgang.
Schließlich erhält man als Repräsentantensystem $E_\Xi = \left\{ \begin{pmatrix} -1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$

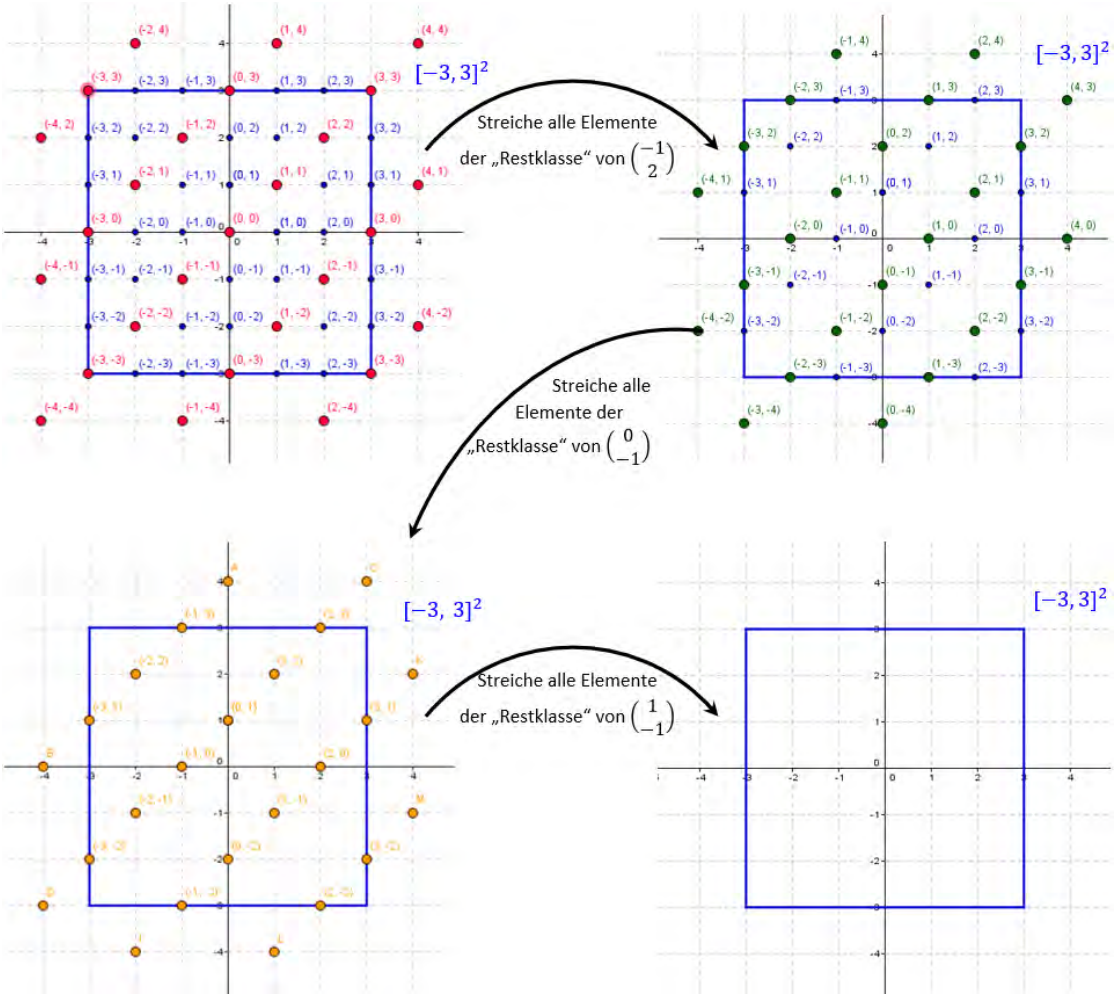


Abbildung 2.20: Die andersfarbig markierten Elemente gehören jeweils zu einer „Restklasse“und werden pro Durchlauf gestrichen, während eines davon zum Repräsentantensystem hinzugefügt wird.

Bemerkung 2.4.12
Ein Repräsentantensystem einer Quotientengruppe $\mathbb{Z}/\Xi\mathbb{Z}$ lässt sich durch geschicktes Ablesen auch einfacher bestimmen.
Betrachtet man dazu das halboffene Einheitsquadrat $[0,1)^2$ bzw. im eindimensionalen Fall das Einheitsintervall $[0,1)$ und dessen Bild unter der Matrix Ξ so lassen sich die Repräsentanten wie folgt schreiben: $E_\Xi = \Xi \cdot [0,1)^s \cap \mathbb{Z}^s$ für $s \in \{1,2\}$.

Einfach gesprochen verformt die Matrix das Einheitsquadrat bzw. das Einheitsintervall wie wir bereits wissen und man nimmt alle ganzen Zahlen/Vektoren, die in jenem verformten Objekt liegen und packt sie in ein Repräsentantensystem.

Beispiel 2.4.13 Ablesen der Repräsentanten einer Quotientengruppe

- Für das eindimensionale Beispiel $\mathbb{Z}/3\mathbb{Z}$ lässt sich das triviale Repräsentantensystem $E_3 = \{0, 1, 2\}$ ablesen (siehe Abbildung 2.21).

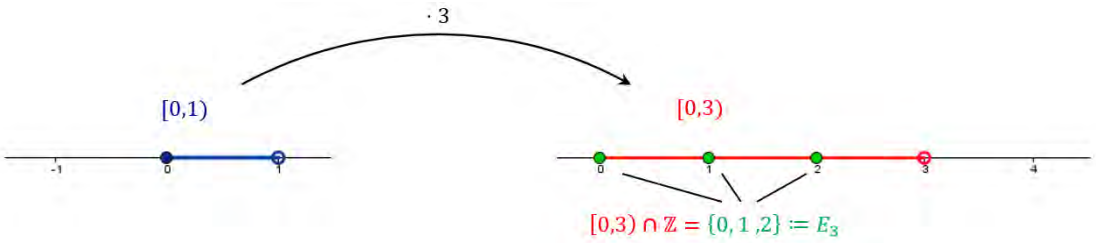


Abbildung 2.21: Das halboffene Einheitsintervall $[0, 1)$ wird durch die (1×1)-Matrix 3 zu $[0, 3)$. Die Schnittmenge dieses Intervalls mit der Menge der ganzen Zahlen $\mathbb{Z}$ sollte nicht allzu schwer zu bilden sein.

- Für das zweidimensionale Beispiel $\mathbb{Z}/\Xi\mathbb{Z}$ mit $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ lässt sich das triviale Repräsentantensystem $E_\Xi = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$ ablesen (siehe Abbildung 2.22).

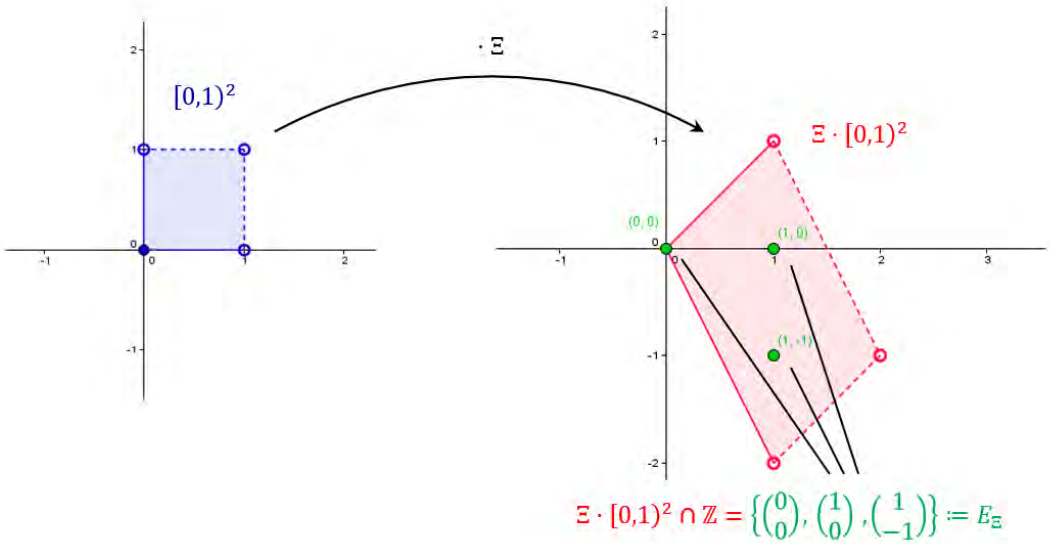


Abbildung 2.22: Das halboffene Einheitsquadrat $[0,1)^2$ wird durch die (2×2)-Matrix $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ zu einem Parallelogramm verformt. Die Schnittmenge dieses Parallelogramms mit $\mathbb{Z}^2$ kann direkt abgelesen werden.

Im weiteren Verlauf wird als Repräsentantensystem für $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ durchgehend das soeben Angegebene verwendet.

Bemerkung 2.4.14 Eckpunkte des verformten halboffenen Einheitsquadrats
Durch die Anwendung der Matrix Ξ auf das halboffene Einheitsquadrat $[0,1)$ wird dieses verzerrt bzw. verformt. Egal wie Ξ aufgebaut ist, der Ursprung $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ bleibt in der verformten Fläche enthalten. Alle weiteren Eckpunkte der verzerrten Fläche sind nicht in ihr enthalten, da das ursprüngliche Einheitsquadrat halboffen ist. Diese nicht in der Fläche enthaltenen Eckpunkte gehören alle zu der Äquivalenzklasse des $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$-Vektors.

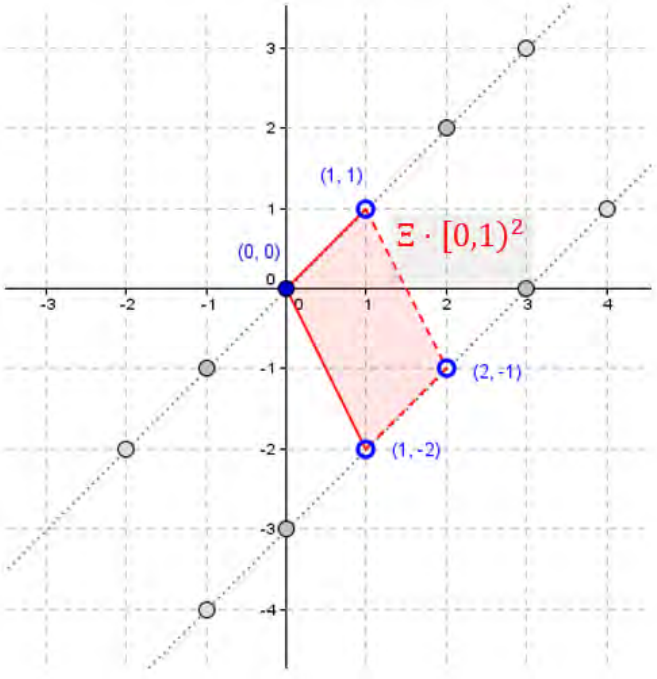


Abbildung 2.23: Das halboffene Einheitsquadrat $[0,1)^2$ wird durch die (2×2)-Matrix $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ zu einem Parallelogramm verformt. Die Eckpunkte dieses Parallelogramms gehören alle zur Äquivalenzklasse von $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$. Legt man gedanklich Geraden durch diese Eckpunkte, so findet man leicht weitere Repräsentanten aus dieser Äquivalenzklasse.

2.4.3 Filter

Definition 2.4.15 LTI-Filter [4]
Ein **linearer zeitinvarianter Filter** (**L**inear **T**imeinvariant **F**ilter) oder kurz **Filter** ist eine Abbildung, die jedem Eingangssignal ein eindeutig bestimmtes Ausgangssignal unter Berücksichtigung folgender Eigenschaften zuordnet:

(i) Zeitinvarianz: Ein gegenüber dem ursprünglichen Signal verschobenes Eingangssignal soll zu einem entsprechend verschobenen Ausgangssignal führen.

(ii) Linearität: Das Ausgangssignal soll linear vom Eingangssignal abhängen, d.h. Summe und Vielfaches vom Eingangssignal führt entsprechend zu Summe und Vielfachem vom Ausgangssignal.

Definition 2.4.16 Impulsantwort [4]
Die Impulsantwort f eines Filters beschreibt das Ausgangssignal des Filters, wenn am Eingang der Einheitsimpuls $\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$ (auch Dirac-Impuls) anliegt.

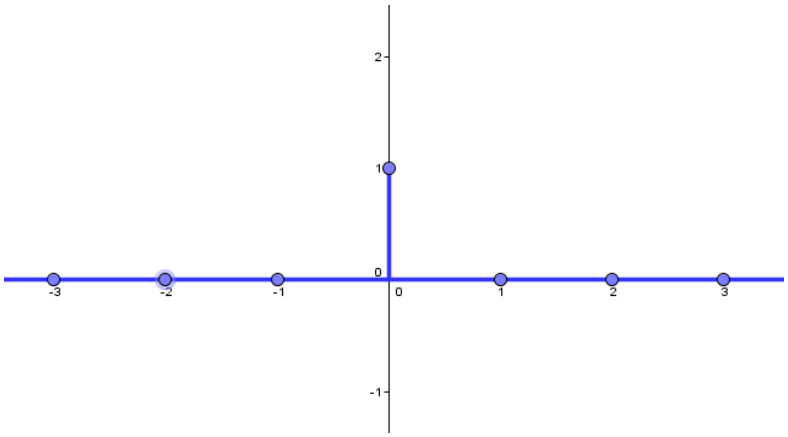


Abbildung 2.24: Einheitsimpuls oder Dirac-Impuls

Definition 2.4.17 FIR-Filter [4]
Ein FIR-Filter (**F**inite **I**mpulse **R**esponse) besitzt eine endliche Impulsantwort, d.h. das Ausgangssignal des Filters zum Einheitsimpuls ist nur an endlich vielen $n \in \mathbb{Z}$ ungleich Null. Fast alle $f(n)$ verschwinden.

Satz 2.4.18 Faltung mit der Impulsantwort [4]
Ein einfacher Weg zur Berechnung des Ausgangssignals y eines Filters ist die Faltung des Eingangssignals x mit der Impulsantwort f des Filters.

$$y(n) = f * x = \sum_{k=-\infty}^{\infty} f(k) \cdot x(n-k) \tag{2.1}$$

Bemerkung 2.4.19
Das Ausgangssignal eines FIR-Filters als Faltung mit der endlichen Impulsantwort besteht aus endlich vielen Summanden.

2.4.4 Filterbänke

Beim Downsampling werden bestimmte Abtastwerte eines Signals entfernt, beim Upsampling werden zwischen einzelnen Abtastwerten eines Signals Nullwerte eingefügt.

Definition 2.4.20 Downsampling [4]
Beim Downsampling bzw. Unterabtasten eines Signals um den Faktor L wird nur jeder

L-te Wert des ursprünglichen Signals behalten. Alle anderen Werte dazwischen werden verworfen. Das Signal wird ausgedünnt.

Bemerkung 2.4.21
Downsampling wird vor allem beim Analyseteil von Filterbänken verwendet, da sich hier die Datenmenge bei Nutzung von k Filtern um den Faktor $k+1$ vervielfacht. Behält man von den gefilterten Daten nur noch jeden $(k+1)$-ten Abtastwert, so verändert sich die Größe der Daten nicht. Trotzdem kann das Signal aus den einzelnen gefilterten Daten wieder vollständig hergestellt werden (Perfect Reconstruction Filterbank).
Downsampling um den Faktor L wird formal mit $(\downarrow L)$ bezeichnet.

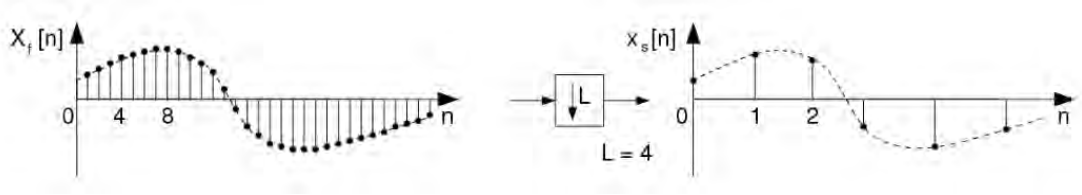


Abbildung 2.25: Das Signal $x_f[n]$ wird um den Faktor 4 unterabgetastet, d.h. nur jeder vierte Wert wird behalten.

Definition 2.4.22 Upsampling [4]
Beim Upsampling eines Signals um den Faktor L werden zwischen je zwei Signalwerten $L-1$ Nullwerte eingefügt. Das Signal wird aufgebläht.

Bemerkung 2.4.23
Upsampling wird vor allem beim Syntheseteil von Filterbänken verwendet, da zuvor bereits ausgedünnte Signale wieder zum ursprünglichen Signal zusammengefügt werden sollen. Um alle gefilterten Signale zum Ausgangssignal im Syntheseteil zu addieren, werden die downgesampleten Daten durch Einfügen von Nullen auf die ursprüngliche Größe gebracht.
Upsampling um den Faktor L wird formal mit $(\uparrow L)$ bezeichnet.

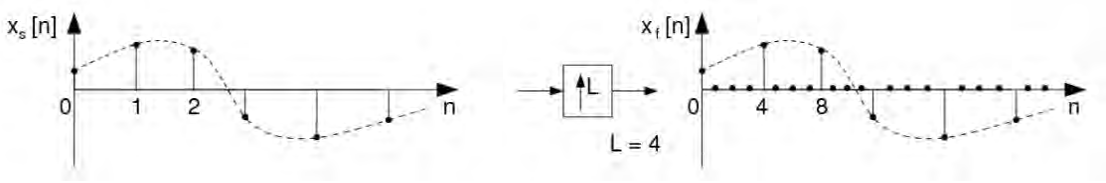


Abbildung 2.26: Das Signal $x_s[n]$ wird um den Faktor 4 upgesamplet, d.h. zwischen je zwei Signalwerte werden $(4-1)$ Nullwerte eingefügt.

Definition 2.4.24 Filterbank [5]

1. Eine **Analyse-Filterbank** besteht aus L einzelnen Analysefiltern und zerlegt ein Eingangssignal x in L Signalkomponenten (auch Teilbänder), welche üblicherweise eine geringere Bandweite und kleinere Abtastraten (Downsampling) besitzen.

2. Eine **Synthese-Filterbank** besteht aus L einzelnen Synthesefiltern und rekonstruiert aus L Teilbändern, in welche das Eingangssignal x durch eine korrespondierende Analyse-Filterbank zerlegt worden ist, das Signal x'.

3. Eine **PR-Filterbank** (Perfect Reconstruction Filterbank) besteht aus einem Analyseteil und einem Syntheseteil. Das Signal x wird im Analyseteil in einzelne Teilbänder zerlegt und im Syntheseteil wieder zu einem Signal x' zusammengefügt. Die PR-Filterbank hat keinen Informationsverlust, es gilt $x = x'$.

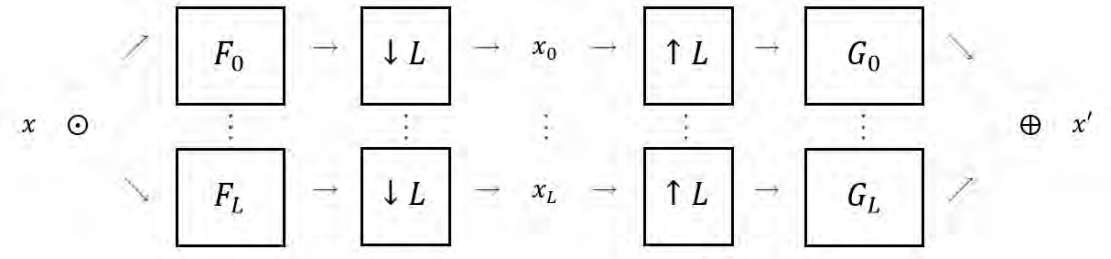


Abbildung 2.27: Das Signal x wird von den Analysefiltern $F_0, ..., F_L$ in L verschiedene Teilbänder zerlegt und jedes einzelne Teilband wird mit Faktor L unterabgetastet. Man erhält L gefilterte Teilsignale. Mit Hilfe von Upsampling und L Synthesefiltern wird aus den Signalkomponenten das Signal x' erzeugt.

Bemerkung 2.4.25
Eine Filterbank mit L verschiedenen Filtern besteht in der Regel aus einem Tiefpassfilter, welcher den Trend aufzeigt, und $L - 1$ Hochpassfiltern, die verschiedene Details hervorheben. Im einfachsten Fall besitzt die Filterbank genau einen Tiefpassfilter und einen Hochpassfilter wie im nachfolgenden Beispiel.

Definition 2.4.26 n-Level Filterbank [5]
Wendet man eine Anzahl an Filtern erneut auf bereits gefilterte Daten an, wird die zugehörige Filterbank 2-Level Filterbank genannt. Wird dieser Prozess n-mal wiederholt, erhält man eine n-Level Filterbank.

Bemerkung 2.4.27

1. Bei wiederholter Filterung durch eine asymmetrische Filterbank werden in jedem Schritt die Details der Hochpassfilter behalten und die einzelnen Filter allein auf die Ausgabe des Tiefpassfilters angewandt.

2. Durch Verwendung von Downsampling reicht zur Speicherung der gefilterten Daten ein Vektor der Länge des Eingangssignals, welcher rekursiv unterteilt wird.

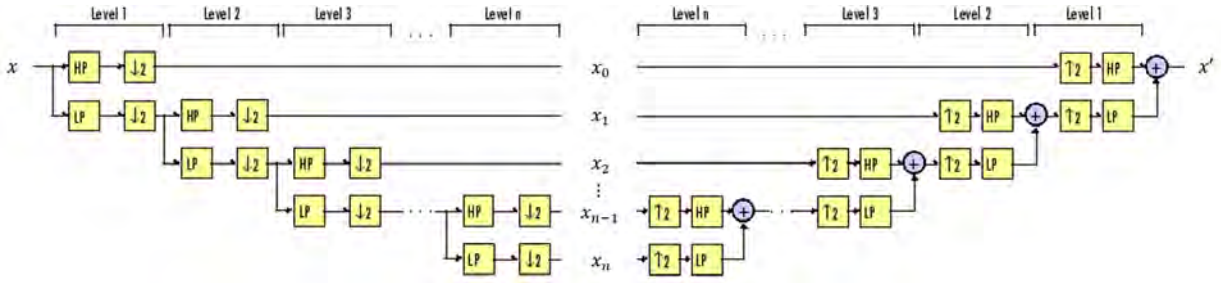


Abbildung 2.28: Exemplarisch wird hier eine sehr einfache Filterbank mit nur einem Tiefpassfilter und einem Hochpassfilter dargestellt. Bei einer asymmetrischen n-Level Filterbank werden die Filter zuerst auf das Eingangssignal x und dann rekursiv $(n-1)$-mal auf die resultieren Tiefpassteilbänder angewandt. In n Teilschritten kann aus den einzelnen Teilbändern durch Addition der Hoch- und Tiefpassdaten wieder ein einziges Signal x' rekonstruiert werden.

Beispiel 2.4.28 2-Level PR-Filterbank
Betrachte eine einfache PR-Filterbank mit einem Tiefpassfilter und einem Hochpassfilter. Im Tiefpassfilter wird der Mittelwert zweier aufeinanderfolgender Signalwerte gebildet und im Hochpassfilter wird die Differenz zweier aufeinanderfolgender Signalwerte gebildet. Der Filter zum gleitenden Mittelwert glättet das Signal, wohingegen der Filter zur gleitenden Differenz die Details erhält.

Filter und deren Impulsantworten für den **Analyseteil** der Filterbank:

- Gleitender Mittelwert: $F_{LP}(x_n) = \frac{x_n + x_{n+1}}{\sqrt{2}}$ mit $f_{LP}(n) = \begin{cases} \frac{1}{\sqrt{2}}, & n \in \{-1, 0\} \\ 0, & sonst \end{cases}$
- Gleitende Differenz: $F_{HP}(x_n) = \frac{x_n - x_{n+1}}{\sqrt{2}}$ mit $f_{HP}(n) = \begin{cases} -\frac{1}{\sqrt{2}}, & n = -1 \\ \frac{1}{\sqrt{2}}, & n = 0 \\ 0, & sonst \end{cases}$

Filter und deren Impulsantworten für den **Syntheseteil** der Filterbank:

- Gleitender Mittelwert: $G_{LP}(x_n) = \frac{x_n + x_{n-1}}{\sqrt{2}}$ mit $g_{LP}(n) = \begin{cases} \frac{1}{\sqrt{2}}, & n \in \{0,1\} \\ 0, & sonst \end{cases}$
- Gleitende Differenz: $G_{HP}(x_n) = \frac{x_n - x_{n-1}}{\sqrt{2}}$ mit $g_{HP}(n) = \begin{cases} \frac{1}{\sqrt{2}}, & n = 0 \\ -\frac{1}{\sqrt{2}}, & n = 1 \\ 0, & sonst \end{cases}$

Betrachte das Signal $x = [1, 3, -1, -1, 2, -1, -3, 1]$ und falte es mit den Impulsantworten der beiden Filter:

- Gleitender Mittelwert: $x * f_{LP} = \left[\frac{\sqrt{2}}{2}, 2\sqrt{2}, \sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -2\sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}\right]$
- Gleitende Differenz: $x * f_{HP} = \left[-\frac{\sqrt{2}}{2}, -\sqrt{2}, 2\sqrt{2}, 0, -\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{2}, \sqrt{2}, -2\sqrt{2}, \frac{\sqrt{2}}{2}\right]$

In beiden Signalkomponenten zusammen steckt nun doppelt so viel Information wie im Ausgangssignal. Beide Teilsignale werden mit Faktor 2 unterabgetastet:

- Gleitender Mittelwert: $(\downarrow 2)(x * f_{LP}) = \left[2\sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}, -\sqrt{2}\right]$
- Gleitende Differenz $(\downarrow 2)(x * f_{HP}) = \left[-\sqrt{2}, 0, \frac{3\sqrt{2}}{2}, -2\sqrt{2}\right] := x_0$

Wende auf den unterabgetasteten gleitenden Mittelwert erneut die beiden Filter an:

- Gleitender Mittelwert: $f_{LP} * ((\downarrow 2)(x * f_{LP})) = \left[2, 1, -\frac{1}{2}, -\frac{1}{2}, -1\right]$
- Gleitende Differenz: $f_{HP} * ((\downarrow 2)(x * f_{LP})) = \left[-2, 3, -\frac{3}{2}, \frac{3}{2}, -1\right]$

Beide Teilbänder müssen nun wiederum einem Downsampling um den Faktor 2 unterzogen werden und man erhält:

- Gleitender Mittelwert: $(\downarrow 2)(f_{LP} * ((\downarrow 2)(x * f_{LP}))) = \left[1, -\frac{1}{2}\right] := x_1$
- Gleitende Differenz: $(\downarrow 2)(f_{HP} * ((\downarrow 2)(x * f_{LP}))) = \left[3, \frac{3}{2}\right] := x_2$

Der Analyseteil der Filterbank ist nun abgeschlossen. Man erhält für die Teilbänder:

- $x_0 = \left[-\sqrt{2}, 0, \frac{3\sqrt{2}}{2}, -2\sqrt{2}\right]$
- $x_1 = \left[1, -\frac{1}{2}\right]$
- $x_2 = \left[3, \frac{3}{2}\right]$

Sämtliche Teilbänder ließen sich nun mit entsprechender Unterteilung in einem Vektor der Länge des Eingangssignals abspeichern.

Beginnen wir mit dem Syntheseteil der Filterbank. Die Teilbänder x_1 und x_2 müssen einem Upsampling mit Faktor 2 unterzogen werden:

- Gleitender Mittelwert: $(\uparrow 2)x_1 = \left[1, 0, -\frac{1}{2}, 0\right]$
- Gleitende Differenz: $(\uparrow 2)x_2 = \left[3, 0, \frac{3}{2}, 0\right]$

Nun wird jeweils der entsprechende Synthesefilter angewandt.

- Gleitender Mittelwert: $g_{LP} * ((\uparrow 2)x_1) = \left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{4}, -\frac{\sqrt{2}}{4}\right]$
- Gleitende Differenz: $g_{HP} * ((\uparrow 2)x_2) = \left[\frac{3\sqrt{2}}{2}, -\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{4}, -\frac{3\sqrt{2}}{4}\right]$

Wir setzen unser ursprüngliches Signal wieder zusammen, indem wir beide Teilbänder addieren. Das Ergebnis stimmt natürlich mit Tiefpassband des ersten Filterdurchlaufs überein.
$g_{LP} * ((\uparrow 2)x_1) + g_{HP} * ((\uparrow 2)x_2) = \left[2\sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}, -\sqrt{2}\right]$
Nun benötigen wir zudem das Hochpassband aus dem ersten Filterdurchlauf und führen erneut ein Upsampling mit Faktor 2 durch.

- Gleitender Mittelwert: $(\uparrow 2)\left[2\sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}, -\sqrt{2}\right] = \left[2\sqrt{2}, 0, -\sqrt{2}, 0, \frac{\sqrt{2}}{2}, 0, -\sqrt{2}, 0\right]$
- Gleitende Differenz: $(\uparrow 2)\left[-\sqrt{2}, 0, \frac{3\sqrt{2}}{2}, -2\sqrt{2}\right] = \left[-\sqrt{2}, 0, 0, 0, \frac{3\sqrt{2}}{2}, 0, -2\sqrt{2}, 0\right]$

Erneut wenden wir die beiden Synthesefilter auf die Teilbänder an:

- Gleitender Mittelwert: $g_{LP} * \left((\uparrow 2)\left[2\sqrt{2}, -\sqrt{2}, \frac{\sqrt{2}}{2}, -\sqrt{2}\right]\right) = \left[2, 2, -1, -1, \frac{1}{2}, \frac{1}{2}, -1, -1\right]$
- Gleitende Differenz: $g_{HP} * \left((\uparrow 2)\left[-\sqrt{2}, 0, \frac{3\sqrt{2}}{2}, -2\sqrt{2}\right]\right) = \left[-1, 1, 0, 0, \frac{3}{2}, -\frac{3}{2}, -2, 2\right]$

Schließlich ergibt die Summe der beiden Signale das Eingangssignal, welches diese PR-Filterbank einmal durchlaufen hat.
$\left[2, 2, -1, -1, \frac{1}{2}, \frac{1}{2}, -1, -1\right] + \left[-1, 1, 0, 0, \frac{3}{2}, -\frac{3}{2}, -2, 2\right] = [1, 3, -1, -1, 2, -1, -3, 1] = x$

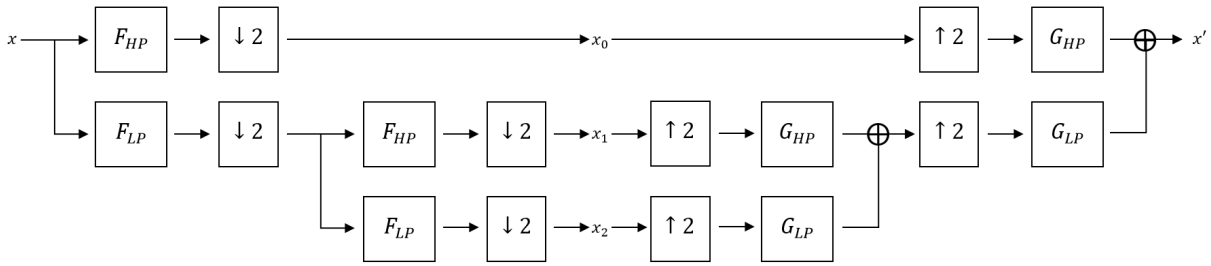


Abbildung 2.29: Asymmetrische 2-Level dyadische Filterbank mit Analyseteil und Syntheseteil.

Bemerkung 2.4.29 Konstruktion einer PR-Filterbank [1]
Zu einer gegebenen expandierenden Matrix [1] lassen sich mit Hilfe von $a = \left(..., 0, \frac{1}{3}, \frac{2}{3}, 1, \frac{2}{3}, \frac{1}{3}, 0, ...\right)$ nach dem folgenden Schema Analysefilter f_ξ und Synthesefilter g_ξ konstruieren, welche zusammen eine PR-Filterbank bilden. Das fettgedruckte Element stellt jeweils das $(0,0)$-**Element** dar.

- Tiefpassfilter: $f_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ und $g_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{2}{3} & \mathbf{1} & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
- Hochpassfilter: Hier werden die Repräsentanten der Quotientengruppe $\mathbb{Z}/\Xi\mathbb{Z}$ benötigt. Das $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$-Element, welches per Definition im Repräsentantensystem liegt, ist dem bereits behandelten Tiefpassfilter zugeordnet. Nun wird für jedes weitere $\xi_j \in E_\Xi \backslash \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$ ein Hochpassfilter erzeugt. Dessen $(0,0)$-Element ist um den Vektor ξ_j verschobenen.

Für die genaue mathematische Berechnungsmethode einer Perfect Reconstruction Filterbank siehe [1].

Beispiel 2.4.30 Konstruktion einer PR-Filterbank zu einer gegebenen Matrix
Sei $\Xi = \begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$ eine expandierende Matrix mit $det(\Xi) = -3$. Die Quotientengruppe $\mathbb{Z}/\Xi\mathbb{Z}$ wird durch $E_\Xi = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$ repräsentiert. Abbildung 2.30 zeigt die konstruierten Filter.

[1] Eine Matrix $\Xi \in \mathbb{Z}^{x \times s}$ heißt expandierende Matrix, wenn all ihre Eigenwerte im Absolutbetrag strikt größer als 1 sind.

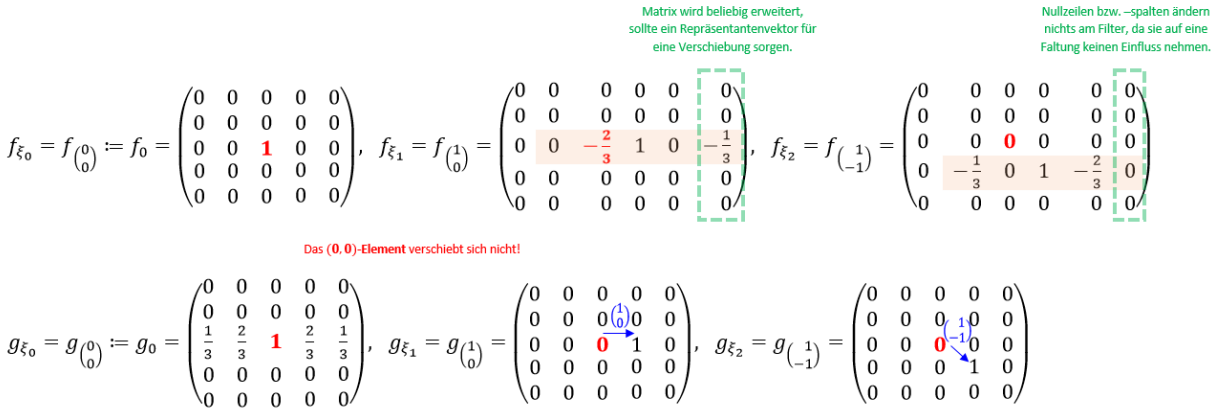


Abbildung 2.30: Zu der gegebenen Matrix Ξ wurde eine PR-Filterbank nach angegebenem Schema konstruiert.

3 Anhang

3.1 MatLab Erklärungen

3.1.1 Darstellung der Impulsantwort eines Filters

Ein FIR-Filter lässt sich am einfachsten durch seine Impulsantwort darstellen. Wie der Name schon sagt, besitzt ein FIR-Filter eine endliche Impulsantwort $f(n)$, d.h. es gilt $f(n) = 0$ für fast alle $n \in \mathbb{Z}$. In MatLab wird ein Filter anhand seiner Impulsantwort durch ein zweidimensionales Array $[a, n]$ dargestellt, wobei für jede Amplitude $a \neq 0$ das zugehörige Sample n gespeichert wird.
Für den Filter $A(x)_n = \frac{x_n + x_(n+1)}{\sqrt{(2)}}$, welcher den gleitenden Durchschnitt eines diskreten Signals $(x_n)_(n \in \mathbb{Z})$ bildet, wird seine Antwort auf den Einheitsimpuls wie folgt dargestellt:

$$[a, n] = \begin{bmatrix} \frac{1}{\sqrt{2}} & -1 \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

3.1.2 Smith-Faktorisierung

Die Smith-Faktorisierung einer Matrix A lässt sich in MatLab mit Hilfe der Funktion *SmithFactorization.m* wie folgt berechnen:

```
>> C = [1, 1; 1, -2]

C =

    1     1
    1    -2

>> [U, D, V] = smith(C)

U =

    4     1
    1     0

D =

    1     0
```

```
    0     3

V =

    1    -2
   -1     3
```

3.1.3 Downsampling und Upsampling

Sampling von eindimensionalen Daten

Eindimensionale Signale können in MatLab recht einfach einem Down- bzw. Upsampling unterzogen werden. Gegeben sei das Signal $x = [1, 3, -1, -1, 2, -1, -3, 1]$.

```
>> x = [1, 3, -1, -1, 2, -1, -3, 1]

x =

    1     3    -1    -1     2    -1    -3     1

>> down2 = x(1:2:end)

down2 =

    1    -1     2    -3

>> up2 = zeros(1, 2*length(x));
>> up2(1:2:end) = x

up2 =

Columns 1 through 12

    1     0     3     0    -1     0    -1     0     2     0    -1     0

Columns 13 through 16

    -3     0     1     0
```

Sampling von zweidimensionalen Daten

Zweidimensionale Daten wie zum Beispiel Bilder können in MatLab gesamplet werden, indem nacheinander in beide Dimensionen das Down- bzw. Upsampling angewandt wird.

Gegeben sei die Matrix $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$.

Die Matrix soll in beide Dimensionen um den Faktor 2 unterabgetastet werden. Mit der resultierenden Matrix wird schließlich ein Upsampling um den Faktor 3 in die eine Dimension und um den Faktor 2 in die andere Dimension durchgeführt.

```
>> A = [ 1,  2,  3,  4; ...
         5,  6,  7,  8; ...
         9, 10, 11, 12; ...
        13, 14, 15, 16]

A =

    1     2     3     4
    5     6     7     8
    9    10    11    12
   13    14    15    16

>> down21 = A(1:2:end,:)

down21 =

    1     2     3     4
    9    10    11    12

>> down22 = down21(:,1:2:end)

down22 =

    1     3
    9    11

>> up31 = zeros(3*size(down22,1), size(down22, 2)); up31(1:3:end,:) = down22

up31 =

    1     3
    0     0
    0     0
    9    11
    0     0
    0     0
```

```
>> up32 = zeros(size(up31,1), 2*size(up31, 2)); up32(:,1:2:end) = up31

up32 =

     1     0     3     0
     0     0     0     0
     0     0     0     0
     9     0    11     0
     0     0     0     0
     0     0     0     0
```

3.2 MEX Erklärungen

3.2.1 Speicherart von Matrizen

Die 'Column-Major-Order' und die 'Row-Major-Order' beschreiben Arten der Speicherung von Matrizen in einem linearen Speicher wie etwa dem Arbeitsspeicher. Bei der 'Column-Major-Order' wird eine Matrix spaltenweise abgespeichert, bei der 'Row-Major-Order' zeilenweise.
Folgendes Beispiel soll die Speicherung einer Matrix bezüglich der beiden unterschiedlichen Arten darstellen:

$$M = \begin{pmatrix} 17 & 4 & 11 \\ 3 & 8 & 29 \end{pmatrix}$$

Spaltenweise Abspeicherung

Adresse	**Wert**
0	17
1	3
2	4
3	8
4	11
5	29

Zeilenweise Abspeicherung

Adresse	**Wert**
0	17
1	4
2	11
3	3
4	8
5	29

MatLab speichert Matrizen spaltenweise. Werden Matrizen als Parameter von MatLab an die MEX-Schnittstelle übergeben, so wird ein Zeiger auf den ersten Wert der Matrix im Arbeitsspeicher und die Anzahl ihrer Zeilen und Spalten übergeben. Hier ist darauf zu achten, dass die Matrix im Arbeitsspeicher in 'Column-Major-Order' abgelegt ist.
Nachstehendes Beispiel illustriert die Weitergabe der Matrix M an die MEX-Schnittstelle.

$$M = \begin{pmatrix} 17 & 4 & 11 \\ 3 & 8 & 29 \end{pmatrix} \qquad \left[\begin{array}{c|c} \textbf{Adresse} & \textbf{Wert} \\ \vdots & \vdots \\ 0x00000005 & 17 \\ 0x00000006 & 3 \\ 0x00000007 & 4 \\ 0x00000008 & 8 \\ 0x00000009 & 11 \\ 0x0000000A & 29 \\ \vdots & \vdots \end{array}\right]$$

Abbildung 3.1: In etwa so ist die Matrix M im Arbeitsspeicher abgelegt. Hat man einen Zeiger auf das erste Element, also auf die Adresse $0x00000005$ und kennt die Anzahl der Zeilen und Spalten der Matrix, so kann man auf ihre einzelnen Einträge zugreifen.

Jede MEX-Funktion erhält alle angegebenen Parameter im $*prhs[]$-Array (**P**ointer **R**ight **H**and **S**ide). Der nachstehende Codeschnipsel verdeutlicht die Verarbeitung der übergebenen Parameter in der MEX-Funktion, wobei davon ausgegangen wird, dass sich die Matrix M im ersten Feld des $prhs$-Arrays, also in $prhs[0]$ befindet.

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *pointerToFirstArrayElement = mxGetPr(prhs[0]);
    size_t numberOfArrayRows = mxGetM(prhs[0]);
    size_t numberOfArrayColumns = mxGetN(prhs[0]);
}
```

Abbildungsverzeichnis

2.1 Verzerrung des halboffenen Einheitsquadrats durch eine Transformationsmatrix . . . 7
2.2 Einzelschritte der Verzerrung des halboffenen Einheitsquadrats durch eine Transformationsmatrix . . . 7
2.3 Anwenden eines Verschiebungsvektors auf eine Matrix . . . 9
2.4 Anwenden eines negativen Verschiebungsvektors auf eine Matrix . . . 10
2.5 Downsampling einer Matrix in X- bzw. Y-Richtung . . . 11
2.6 Downsampling eines Testbildes bezüglich einer Streckungsmatrix . . . 13
2.7 Upsampling einer Matrix in X- bzw. Y-Richtung . . . 14
2.8 Upsampling eines Testbildes bezüglich einer Streckungsmatrix . . . 15
2.9 Analyse des Testbildes Lena mit zwei parallelen Filterbänken . . . 17
2.10 Analyse des Testbildes Lena mit zwei parallelen Filterbänken in Baumansicht . . . 18
2.11 Analyse des Testbildes Lena mit einer einzelnen Filterbank . . . 19
2.12 Analyse des Testbildes Lena mit einer einzelnen Filterbank in Baumansicht 20
2.13 Hochpass-Analyse des Testbildes Lena . . . 21
2.14 Rekonstruktion eines Bildes . . . 22
2.15 Upsampling und Faltung von Tief- und Hochpassinformationen . . . 23
2.16 Koeffizientenmatrizen einer Filterbank . . . 24
2.17 Restklassenring mit farbig gekennzeichneten Restklassen . . . 41
2.18 Produkt aus Matrix und einer Fläche . . . 43
2.19 Elemente von Omega . . . 44
2.20 Streichen von Elementen aus Omega . . . 45
2.21 Ablesen eines Repräsentantensystem im Eindimensionalen . . . 46
2.22 Ablesen eines Repräsentantensystem im Zweidimensionalen . . . 47
2.23 Eckpunkte des verformten halboffenen Einheitsquadrats . . . 48
2.24 Einheitsimpuls . . . 49
2.25 Downsampling eines Signals . . . 50
2.26 Upsampling eines Signals . . . 50
2.27 Filterbank . . . 51
2.28 Asymmetrische n-Level dyadische Filterbank . . . 52
2.29 Asymmetrische 2-Level dyadische Filterbank . . . 54
2.30 Konstruierte Filter einer PR-Filterbank . . . 56

3.1 Abspeicherung einer Matrix im Arbeitsspeicher . . . 61

Literaturverzeichnis

[1] Mariantonia Cotronei, Daniele Ghisi, Milvia Rossini, Tomas Sauer; 'An anisotropic directional subdivision and multiresolution', *Advances in Computational Mathematics*, 2014.

[2] Hans-Joachim Kowalsky, Gerhard Michler; 'Lineare Algebra', S. 328 ff., De Gruyter Verlag, 8. Auflage, 1977.

[3] K. Rosenbaum; 'Algebra', S. 16 ff., 2011,
https://www.tu-ilmenau.de/fileadmin/media/dma/rosenbaum/Vorlesung7.pdf.

[4] Werner Bäni; 'Wavelets - Eine Einführung für Ingenieure', Oldenburg Wissenschaftsverlag, 1. Auflage, 2002.

[5] Tomas Sauer; 'Shearlet multiresolution and multiple refinement', In: Gitta Kutyniok 'Shearlets', Springer, 2011.