M2 - TSI

UE34 Laboratory Report

Image & Signal Processing

Authors:
Arthur Scharf
Andreas Wenzel

February 3, 2017

1 Introduction - Image Processing for Earth Observation

Image processing and the analysis of image data plays an important role especially for Earth Observations, but also in other fields related to space applications that make use of images, e.g. optical navigation or optical attitude control systems. In this report, we present a thorough overview of examples for image processing techniques, especially applying examples in Matlab. We focus on colour spaces for images, fourier transforms of basic figures, erosion and delatation and classification.

1.1 Computer Vision

From a mathematical point of view, an image is an application that associates a value $I(u, v)$ to a pixel (u, v). Pixels are atomic elements arranged in rows and columns, such that the size of an image is given by the number of rows and the number of columns. Therefore, an image can be processed by using a matrix representation, where each matrix element (u, v) represents a pixel containing the value $I(u, v)$ of the pixel.. In the case of multispectral images, i.e. colour images like RGB images (see following chapter), an image is an application that associates several values, e.g. $I_1(u, v)$ $I_2(u, v)$ and $I_3(u, v)$ to a pixel (u, v), where the values I_i represent a different colour like red, green and blue in RGB images. Therefore, each matrix element (u, v) contains a vector of values. This matrix notation is the basic of image processing using Matlab.

2 Greyscale and Colour images

The understanding of colour images and the connection to their greyscale representation are fundamental for image analysis and image processing. We illustrate colour images in this section by considering the examples of RGB and HSV colour space. There are also other colour spaces like CYMK (cyan, yellow, magenta, black) or YUV, but we only consider the first two in our examples since these are the most frequently used ones and essential for the understanding of coloured images. CYMK and YUV are mainly based on the first two colour spaces.

2.1 Greyscale Images

Greyscale images are basic images where the value of each pixel e.g. is associated to a value between 0, which is black, to 1 for bright. An example including an optical illusion is presented in image 1. For this image, a 512×1024 matrix was created in Matlab, first being black by associating = to each element, but then was filled with increasing values from 0 to 1 along each line. in the middle of the image, a stripe of constant values 0.5 was created. By just

concentrating on the upper and lower part of the image, one can see how the brightness increases from left to right. Although the middle layer consists of the same greyscale value for the individual pixels, it appears to be brighter in on the left and darker on the right. This is an optical illusion created by the increasing brightness from left to right of the upper and lower part.

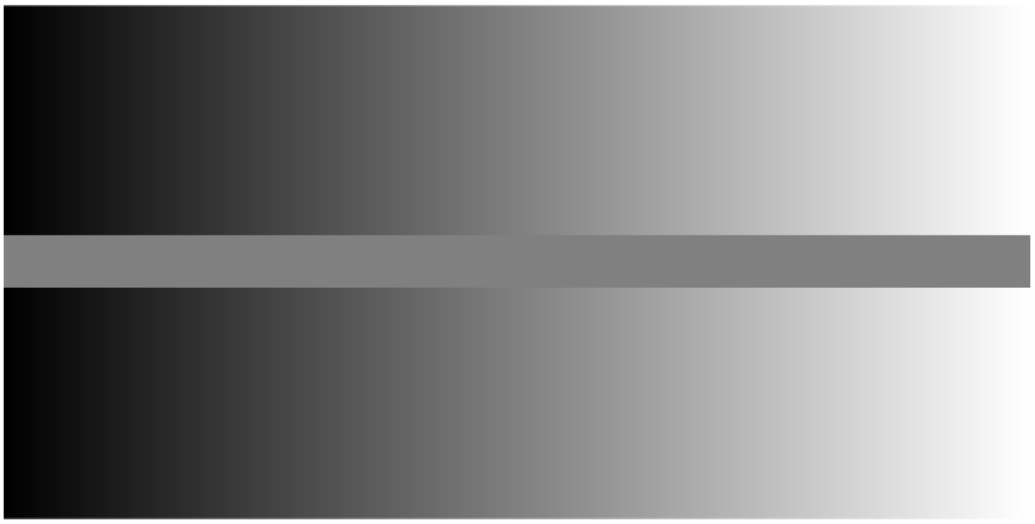

Figure 1: Greyscale image with optical illusion.

2.2 RGB

As mentioned before, in the case of a color image, 3 values are associated to each pixel in an RGB image, representing 8bit unsigned integer values (UINT8) for the colours of red, green and blue. By mixing these values for the colours, one can create any colour for the individual pixel. For the pixel to appear black, all the values of each pixel have to be 0, while they are 1 for a white image, which represents the additive nature of RGB images. As an example, we created the German flag by creating a 300×500 matrix, where the values for each pixel in the first 100 lines are set to 0, to 1 for I_R (red) and 0 for the other two in the lines from 101 to 200, and finally to $I_R = 250$,$I_G = 215$ and $I_B = 0$ in order to achieve a gold colour for the lower lines.

Figure 2: RGB image of the German flag.

2.3 HSV Colour Space

The HSV colour space consists of a value for hue (H), which is a value for the colour or wavelength running from 0 to 1, saturation (S) describing the intensity of that colour also running from 0 to 1, and the value (V) describing the darkness or brightness. To illustrate this, the image 3 was created. A matrix of 512×750 elements was created. For each line, the H value increases from left with 0 to right with 1, resulting in different colours for a spectrum along each line. For each column, the S value varies from 0 at the top of the image to 1 at the bottom. However, for all elements a brightness value of $V = 1$ was chosen.

Figure 3: Illustration of HSV colour space.

Figure 4: Illustration of HSV colour space in a colour wheel.

As an additional illustration, the HSV colour space is represented in the colour wheel shown in figure 4. Inside the wheel, the saturation gradually increases from inside to outside, while the hue value depends on the angle in the circle. Inside the circle, the brightness is chosen as a constant with $V = 1$, while it is set to 0 outside, resulting in the black frame around the circle.

2.4 PAN/XS Fusion

A very important fact regarding multispectral images is that there is a tradeoff between spatial resolution and spetral resolution for every image, also related to the sensors. Monochromatic images only require one channel, and therefore can have a very high spatial resolution while losing spectral information since they are monochromatic. On the other hand, multispectral images need several channels to store the various colour information, which results in a loss of spatial resolution. A very important application of image processing is therefore to fuse monochromatic images of high spatial resolution with multispectral images of low spatial resolution in order to achieve a high resolution multispectral image. An example of that is presented in figures 5 to 7. Figures 5 and 6 show both the initial images, where one can clearly see the difference in spatial resolution between the monochromatic and the hyperspectral image and also the difference in size of the two images. In our algorithm, the latter is resized to the size of the monochromatic one, and then additionally extended by values of the high resolution image. The resulting multispectral image is presented in figure 7. Note, that this image should actually have the same size as the first monochromatic image. One can see that the spatial resolution of the coloured image is improved now. There are a few regions, where the information from the monochromatic image fills up elements that were empty due to the resizing operation.

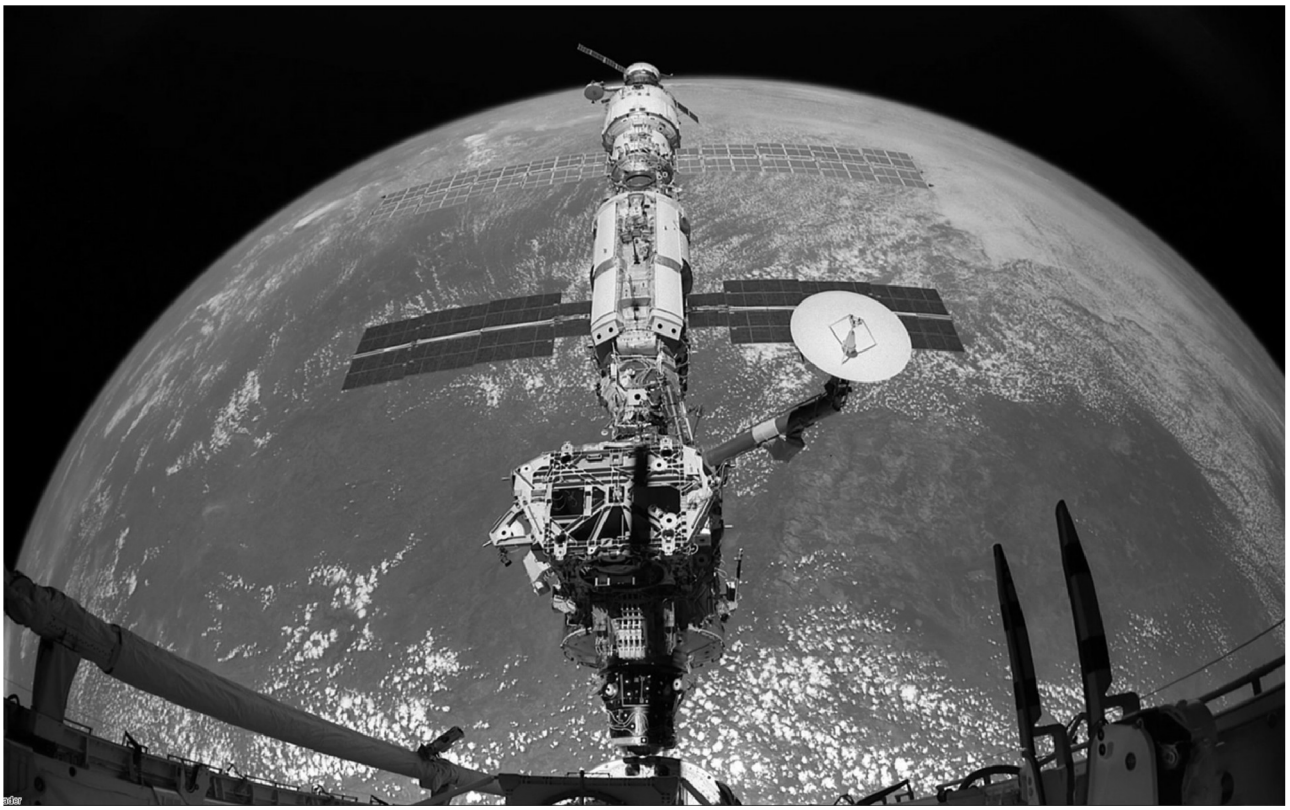

Figure 5: Monochromatic image with high spatial resolution.

Figure 6: Hyperspectral image with low spatial resolution

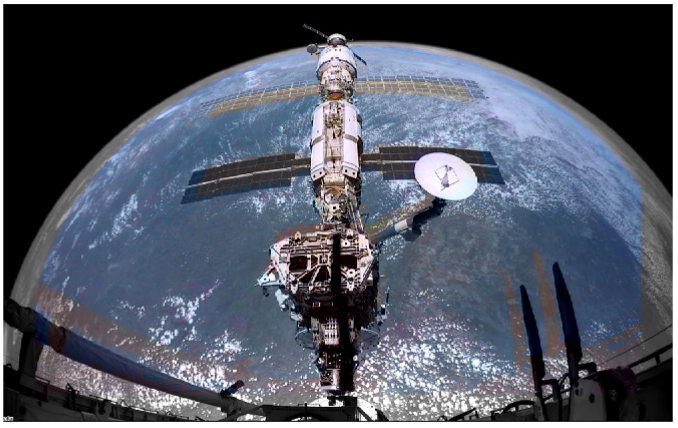

Figure 7: Fused multispectral image with high spatial resolution.

2.5 Image processing and image analysis with Fouriertransform

Fouriertransforms are a very important tool to analyse images. The fourier transform describes a reversible transformation from space domain into the fourier domain given by the following relation:

$$\hat{I}(u,v) = \sum_{u=0}^{U-1} \sum_{v=0}^{V-1} I\left(u,v\right) \cdot e^{-j2\pi\left(\frac{f\cdot u}{U}+\frac{g\cdot v}{V}\right)}$$

As examples, figures 14 to 15 show the fourier transforms of several fundamental shapes. The shape images are created in greyscale by setting every pixel outside the shape itself to 0, while the shape is defined by pixels using the value one 1. The shapes that are investigated are a long band across the image, a square in the centre , a rectangle a the centre, and a circle at the image centre . The fourier transform of the first shows a sinc function along the Y-axis (lines), since the band can be seen as a rectangle function along the line-axis, whereas the fourier transformation is extremely thin along the x axis, since it is a constant (of 0) along the x-axis. Similar to that, a sinc-function along both axis results for the fouriertransform of the square, since the square can be seen as a 2D rectangular function. In analogy, a sinc-function results from the rectangle. However, the sidelobes are more narrow for the y-axis (lines), since the rectangle in the image space is broader along the line-axis. Finally, the fourier transform of the circle are Bessel functions, which are similar to the sinc functions, but show a circular symmetry.

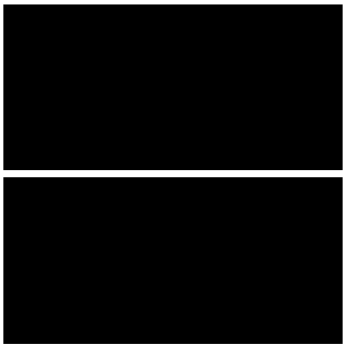

Figure 8: Bar image

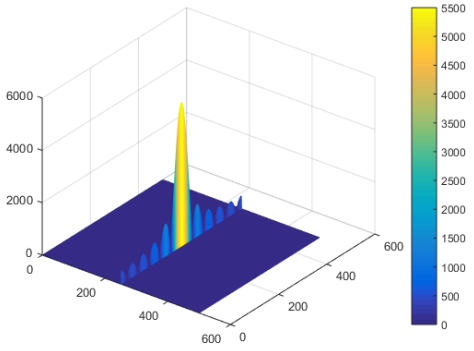

Figure 9: Fourier transform of bar

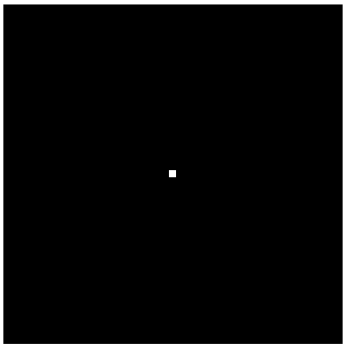

Figure 10: Image of square

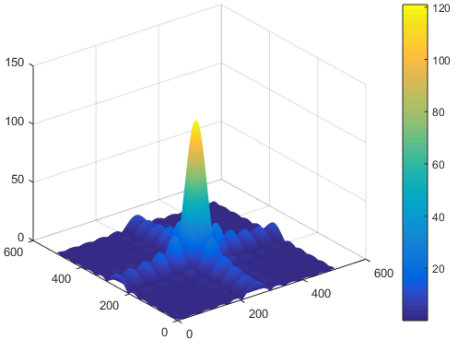

Figure 11: Fourier transform of square

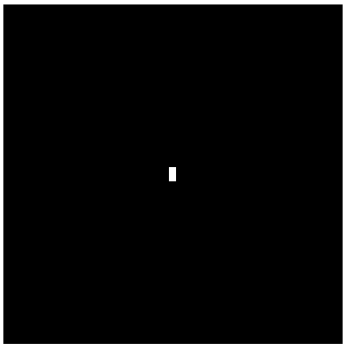

Figure 12: Rectangle image

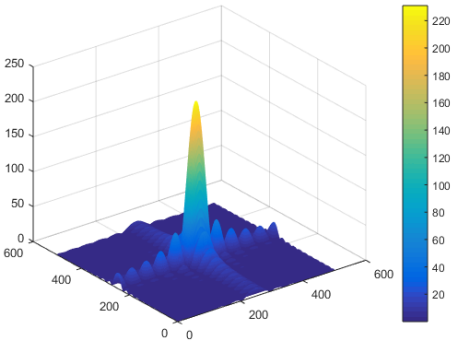

Figure 13: Fourier transform of bar

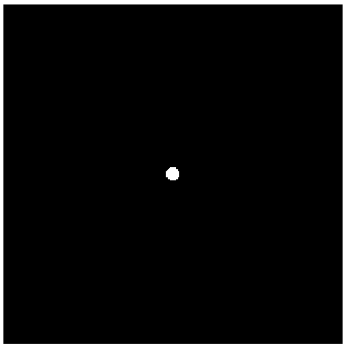

Figure 14: Image of circle

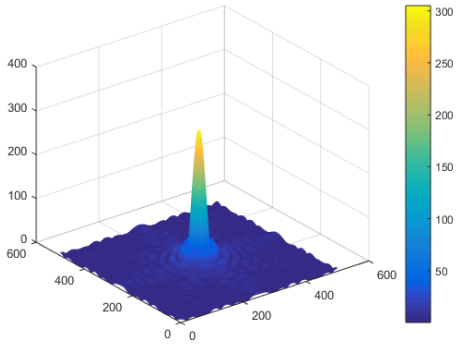

Figure 15: Fourier transform of circle

Understanding the handling between image space and fourier space is useful for image processing. For example, one can identify frequent elements in an image by taking a look at the Fouriertransformation. These frequent elements can then e.g. be filtered out by a filter, such that the processed image only contains a certain element like e.g. a footballfield or certain region of vegetation.

3 Morphology

Morphological operators are applied to detect forms in an image. For this, one can use erosion ϵ and dilatation δ, which both have very different effects on images.

3.1 Erosion & Dilatation

In a mathematical point of view, erosion ϵ is defined as(see [1])

$$I_\epsilon(p) = \epsilon I = \max_{p \in V} I(p)$$

where $I(p)$ is the greyscale image. Therefore, erosion ϵ is the value of p that minimizes $I(p)$ in a given neighbourhood. In analogy, dilatation is defined as:

$$\hat{I}_\delta(p) = \delta I = \min_{p \in V} I(p)$$

[1] source: lecture notes by Emmanuel Zenou

To illustrate the effect of erosion and dilatation, we apply both within Matlab on the image shown in figure 16 . As one can see in figure 17 , erosion smooths the boundaries between neighbouring objects, and therefore reduces the size of white objects in greyscale pictures. In contrast, dilataion reduces black objects in greyscale images, as visible in figure 18, which also results in a smoothening.

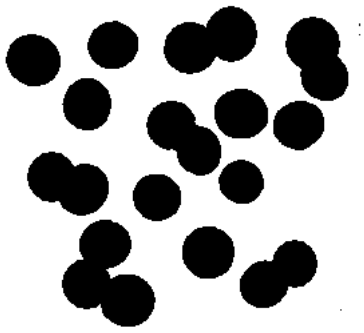

Figure 16: Original image

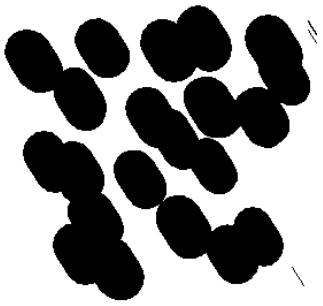

Figure 17: Effect of erosion

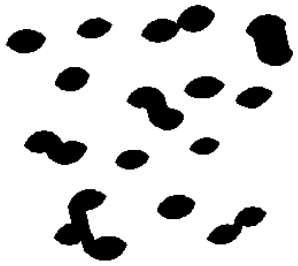

Figure 18: Effect of dilatation

Those effects are also illustrated in figures 19 to 21 applying erosion and dilation to the presented forms. Also a combination of both is shown in figures 22 and 23, where the final result depends on the order in which the operations are applied. Nevertheless, one can see how erosion reduces white edges and thus smooths the image, while dilataion reduces black edges and therefore also smooths the image.

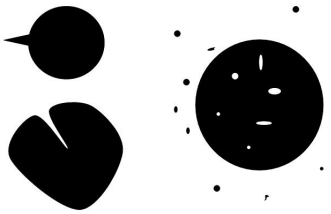

Figure 19: Original image

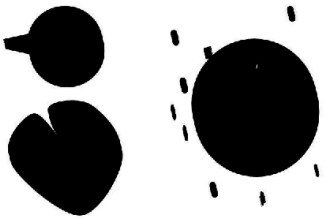

Figure 20: Effect of erosion

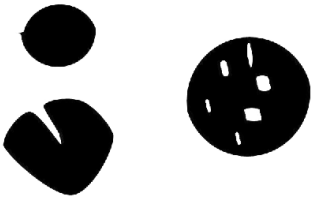

Figure 21: Effect of dilatation

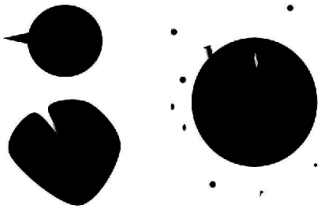

Figure 22: Effect of erosion and dilatation combined

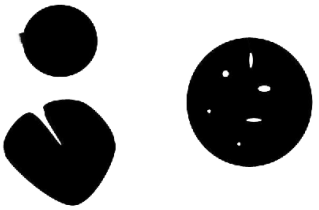

Figure 23: Effect of dilatation an erosion combined

To further illustrate the effect of dilataion, we applied it to a map of Europe shown in figure 24. Then, for that dilated image, the value of each pixel is compared to it's neighbouring one. By progressing this evaluation, a heat heat map results which indicates the distance to the sea (white in the original image), as shown in figure 25.

Figure 24: Original map of Europe.

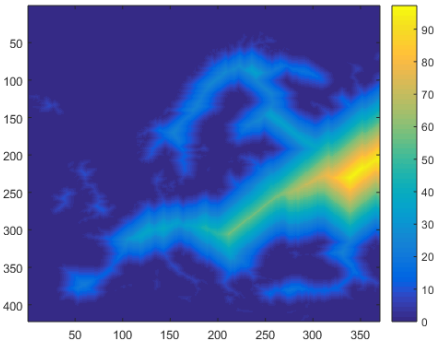

Figure 25: Heat map of Europe showing distances to see after dilatation.

3.2 Morphological Filtering

Morphological Filtering describes a technique that is closely related to operations on the morphology, or shape of structures and features within an image.

Four general filters are the Opening, Closing, Top-Hat and Black-Hat Filters. The morphological opening filter is basically an erosion () followed by a dilatation - which allows to remove small bright spots (spots that are smaller than the structuring element that is defined by the user) and connect dark features. The closing filter is exactly the opposite.

add ref

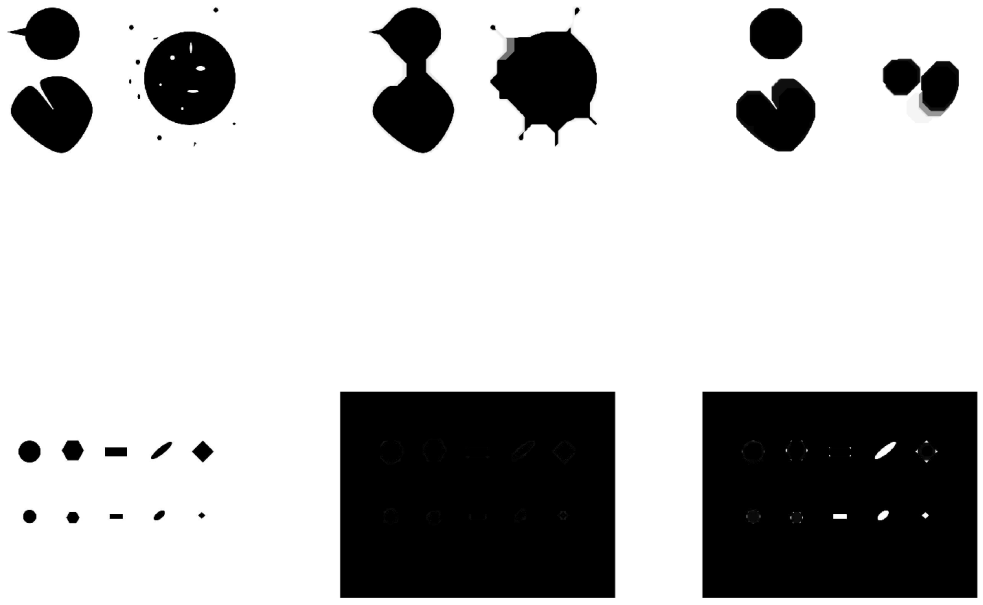

Figure 26: Morphological Filters

4 Data Analysis & Processing

After having extensively discussed Image Processing and Image Analysis in the previous chapters we will now move on to Data Analysis.
In general, Data Analysis is an extension or generalisation of Image Analysis - in particular, an image can also be seen as a set of data with a specific topology. However, data analysis is not limited to images but can be used for a vast amount of applications, be it business, science or others, where a set of data is given and information has to be extracted. Ultimately, this is the goal of data analysis: shaping, modelling and analysing the data to gain information or conclusions from given data. Since this course puts an emphasis on analysing and processing images, all the following examples will be performed on and explained by using images as a data set.

The term *data* itself can be interpreted and defined in multiple ways, which raises the need for a definition in our context of data analysis of images. In general data can be considered as an element taken from a set of data-elements. Each data-element can then be seen as a set of

different components, attributes, parameters etc. - defining what the data-element is composed of - and are often called *descriptors*. Mathematically speaking, this means that our data $\boldsymbol{x}$ is associated to a vector in $\mathbb{R}^{\boldsymbol{n}}$ containing the descriptors,

$$x = (c_1 ... c_n)^T$$

which is also called the *state space* E of our data. These descriptors are the crucial part when analysing data, since they are defining a set of rules according to which we analyse the raw data.

If this terminology is applied to images, the pixels or a range of pixels in each image can be seen as such an descriptor, thus the image itself as our data-element.

4.1 Classification

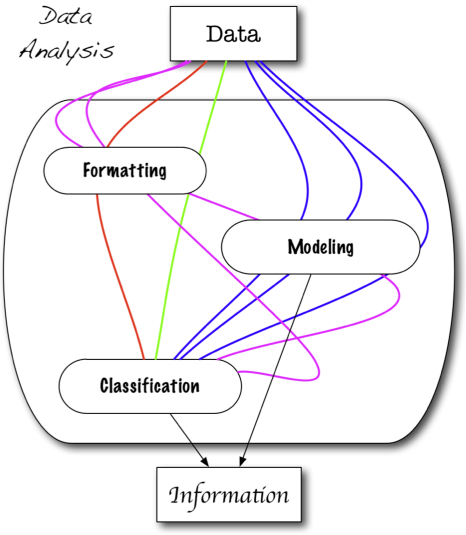

Figure 27: Steps for classification [2]

Having a data set that consists of descriptors essentially allows us to *label* this particular data set. This labelling of data is usually also described as *Classification*, meaning that each data-element or sub-data-element is assigned a particular *class*. In image analysis for space applications typically the pixels in an image have to be classified.

In general, classification of data consists of three steps,

1. Formatting
2. Modelling

[2] source: lecture notes by Emmanuel Zenou

3. Classification

whereas these steps are not compulsory. The formatting step usually consists of finding good descriptors, changing the state space (e.g. by re-shaping), pre-processing data (e.g. filtering) and so on.
The modelling step requires to find a model and its optimal parameters to fit the data, but also to fit the model output to the data and to validate the model. In the final Classification part, the task is to find classes and a classification rule according to which distinct data will be labeled.

There are two main forms of classification, *Supervised* and *Unsupervised*, of which both are described and explained in detail in the following chapters.

4.2 Supervised Classification

Supervised Classification describes a technique, where the classes a data-set contains is known before starting the classification process. This, however, requires a human or another classification algorithm to properly detect and label desired features in either the same or similar images, which implies high effort on pre-processing and preparing the data sets.
For this particular technique there is a great amount of algorithms available, be it kNN, Maximum Likelihood, Bayesian or Linear Classification, Support Vector Machines or Neural Networks. During the laboratory we focused on the kNN algorithm and neural networks in particular, which is why the following sections will provide examples using these techniques.

4.2.1 k - Nearest Neighbour

The k - *Nearest Neighbour* algorithm is the most simple classification algorithm in the supervised classification domain.

The underlying principle is to find the k nearest neighbours to given data, to determine the class of it - which basically results in a calculation of distance. One way to apply this technique to images is to get the desired RGB values of pixels (e.g. by selecting pixels by hand at the desired image feature) and let the algorithm compare the pixels in an image to this particular RGB value.
This can be done of course for a range of RGB values, as is demonstrated in fig. 28, where an image was analysed with respect to three different classes: beach, vegetation and water. For each of these classes RGB values were taken and then used to compare the rest of the image to them. The appropriate MATLAB Code can be found in appendix B.1. Still, it is easy to see that this approach is not perfect. First, it requires to preprocess the image by choosing RGB values. Second, the output images are not perfect in terms of recognising the actual image features we wanted, e.g. when "detecting" water, there are a lot of points recognised on the land-part of the image, since the color is similar.

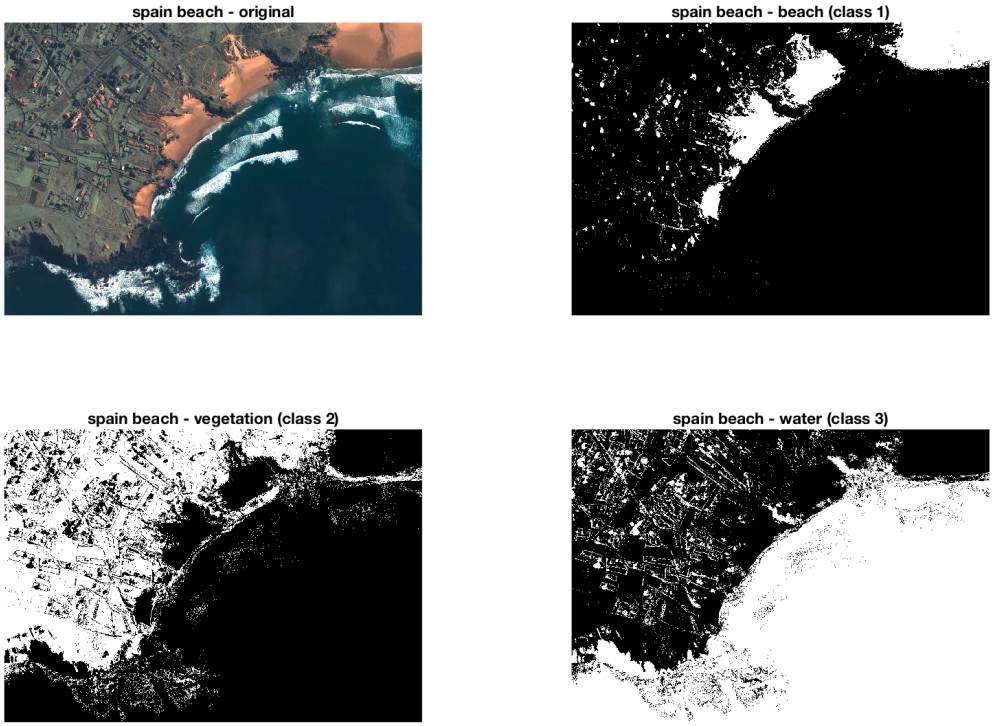

Figure 28: kNN algorithm applied on three classes

4.2.2 Neural Networks

A different approach of classifying data is by using neural networks. Neural Networks are layers of so-called *neurons*, entities that provide a specific output given a specific input. By interconnecting those in- and outputs a network can be created.
But for such a network to be usable, it has to be "trained" - which means that one has to provide input and the desired output for a set of data. According to those given in- and outputs the neurons in the network are assigned a specific "weight". This is done until the desired output for a given input occurs. After that, each neuron has been assigned a weight and the neural network is ready to use.

get the matlab code running for this one

The main advantage of neural networks is that it is very easy to use and can be very efficient. But it also has some drawbacks as one has to provide the desired outcome for input data and the whole process is relying on stochastic.

4.3 Unsupervised Classification

Contrary to Supervised Classification the Unsupervised one does not need a pre-processing of data in terms of classifying example input data. The aim of Unsupervised Classification is to find classes in a given data-set (*Clustering*), model the data (*Regression*) and reduce the state space (*Dimension Reduction*).
For this type of classification there is also a variety of algorithms existing, e.g. the *k-Means* or the *Kohonen Maps* algorithm.

4.3.1 k-Means Algorithm

The k-Means Algorithm is used for Clustering given data into a previously known amount of clusters. It tries to group the data into classes, such as the sum of squared deviations is minimised. Mathematically speaking, this means minimising the cost function

$$J = \sum_{i=1}^{k} \sum_{x_j \in S_i} \left\| x_j - \mu_i \right\|^2$$

with x_j being the distinct data points and μ_i the "Centroid" of a cluster S_i. This results in so-called *Voronoi*-diagrams in the data space - which essentially means that the data points are divided in cells where the centroid is equally spaced with respect to the borders of the cell. Applying this method on the image of the beach in Spain (see section 4.2.1), we can get a result as shown in fig. 29.

One can observe that the clustering worked very well in detecting the sea, vegetation, and beach - even though there are again some smaller errors, e.g. the beach (cluster 3) also contains rooftops of houses nearby, and the waves in the original pictures are assigned to the vegetation (cluster 2) instead of to the water (cluster 1).

4.3.2 Dimension Reduction

If the data to be analysed has a huge i.e. the data dimension is too big (in images this corresponds to large images), one has to reduce the data dimension to reduce the computational amount to find clusters or classes in the data. This is usually done using a technique called *Principal Component Analysis* or in short PCA, where the input data is linearly mapped to new data that has a reduced state space. In practical terms this is usually done by calculating the covariance matrix and its eigenvectors. The eigenvectors corresponding to the largest eigenvalues (also called the principal components) can then be used to reconstruct a great part of the original image.

put in some MATLAB Code and images here for PCA

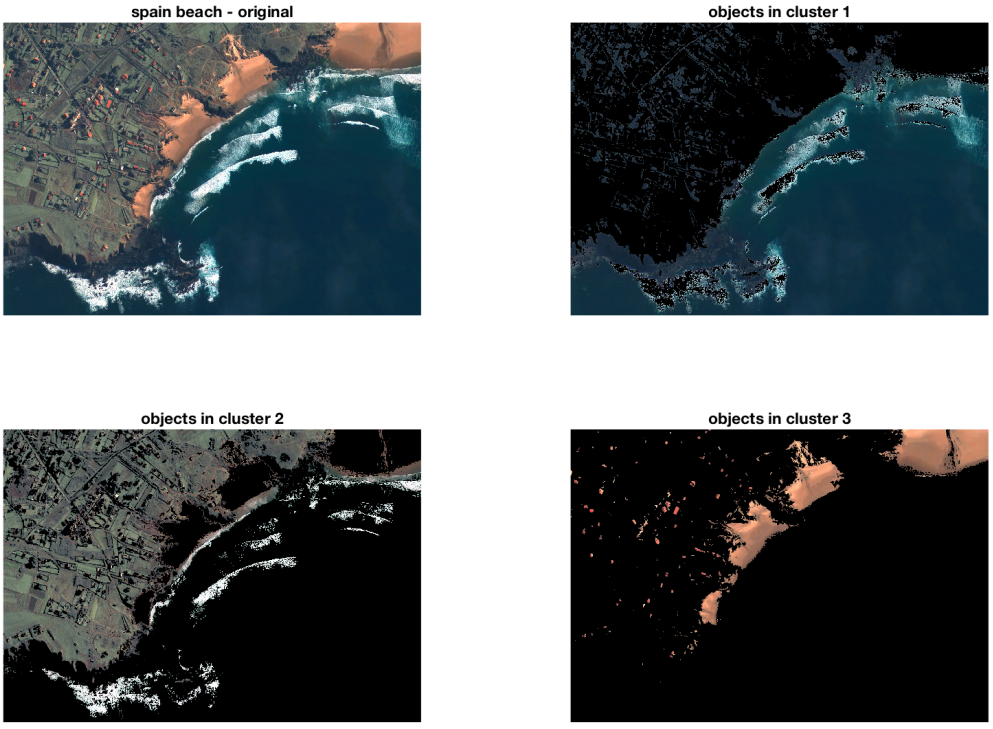

Figure 29: k-Means Algorithm to detect three Clusters of data

5 Conclusion

;

.1 Matlab Code Colourspaces

```
clear all; close all;

Nlines=512;
Ncolumns=1024;
greymatrix=zeros(Nlines,Ncolumns,1);

for k = 1:Nlines
    for i = 1:Ncolumns
        greymatrix(k,i,1)= i/Ncolumns;
    end
end

for k=230:282
    for i = 1:Ncolumns
        greymatrix(k,i,1)= 0.5;
    end
end

figure
imshow(greymatrix)


%%%%Now flags %%%%%%%%%%%%%%%%%%

Nlines=300;
Ncolumns=500;
flag=zeros(Nlines,Ncolumns,3);

for k = 1:100
        flag(k,:,:)=0 ;
end
for k = 100:200
        flag(k,:,1)=1 ;
end
for k = 200:300
        flag(k,:,1)=250 ;
        flag(k,:,2)=215 ;
end


figure
```

```
42 imshow(flag);
43
44 %%%%%%%%%%%%%%%% Do HSV %%%%%%%%%%%%%%%%%%%%%%
45 Nlines=512;
46 Ncolumns=750;
47
48 map=zeros(Nlines,Ncolumns,3);
49
50 for k = 1:Nlines
51     for i = 1:Ncolumns
52         map(k,i,1)= i/Ncolumns;
53         map(k,i,2)= k/Nlines;
54         map(k,i,3) = 1;
55     end
56 end
57
58 map=hsv2rgb(map);
59 figure
60 imshow(map)
61
62 %%%%%%%%%%%%%%%%%%%%Colour wheel %%%%%%%%%%%%%%%%%%%%%%
63
64 Nlines=500;
65 Ncolumns=500;
66
67 totalradius=sqrt((((Nlines/2)-100)^2)+(((Ncolumns/2)-100)^2));
68
69 hsvmap=zeros(Nlines,Ncolumns,3);
70
71 for k= 1:Nlines;
72     for i= 1:Ncolumns;
73
74         R=sqrt(((k-(Nlines/2))^2)+((i-(Ncolumns/2))^2));
75
76         alpha =atan2(((Nlines/2)-k),(i-(Ncolumns/2)));
77
78         if alpha<0
79             alpha=2*pi+alpha;
80         end
81
82         h=alpha/(2*pi);
83         s=R/totalradius;
```

```
84
85          hsvmap(k,i,1)= h;
86          hsvmap(k,i,2)= s;
87          hsvmap(k,i,3) = 1;
88
89          if(R>totalradius)
90          hsvmap(k,i,3) = 0;
91          end
92      end
93  end
94
95
96  map=hsv2rgb(hsvmap);
97  figure
98  imshow(map)
99
100
101 %%%%%%%%%%%%%%%% High res - BW / Low Res - Color %%%%%%%
102
103 close all; clear all;
104 Gre = imread('Ipan_hr.png');
105 Pan = imread('Ihyp_lr.png');
106
107 YCBCR = rgb2ycbcr(Pan);
108
109 YCBCR_res = imresize(YCBCR,[size(Gre,1) size(Gre,2)]);
110
111 YCBCR_res(:,:,1) = Gre(:,:,1);
112
113 result = ycbcr2rgb(YCBCR_res);
114 figure, imshow(result)
```

.2 Matlab Code Fouriertransformation

```
1  clear all; close all;
2
3  Nlines=500;
4  Ncolumns=500;
5  greymatrix=zeros(Nlines,Ncolumns,1);
6
7  for k = 245:255
8          greymatrix(k,:,1)=1.0;
```

```
9  end
10
11 figure
12 imshow(greymatrix)
13
14 FFT=fftshift(fft2(greymatrix));
15 figure
16 shading flat
17 mesh((abs(FFT)));
18 colorbar
19
20 %%%%%%%%%%%%%%%Square %%%%%%%%%%%%%%%%%%%%%%
21
22 Nlines=500;
23 Ncolumns=500;
24 greymatrix=zeros(Nlines,Ncolumns,1);
25
26 for k = 245:255
27     for i = 245:255
28         greymatrix(k,i,1)=1.0;
29     end
30 end
31
32 figure
33 imshow(greymatrix)
34
35 FFT=fftshift(fft2(greymatrix));
36 figure
37 shading flat
38 mesh((abs(FFT)));
39 colorbar
40
41 %%%%%%%%%% Rectangle %%%%%%%%%%%%%%%%
42
43 Nlines=500;
44 Ncolumns=500;
45 greymatrix=zeros(Nlines,Ncolumns,1);
46
47 for k = 240:260
48     for i = 245:255
49         greymatrix(k,i,1)=1.0;
50     end
```

```
end

figure
imshow(greymatrix)

FFT=fftshift(fft2(greymatrix));
figure
shading flat
mesh((abs(FFT)));
colorbar

%%%%%%%%% Circle %%%%%%%

Nlines=500;
Ncolumns=500;
greymatrix=zeros(Nlines,Ncolumns,1);

totalradius=10;

for k = 1:Nlines
    for i = 1:Ncolumns
        R=sqrt(((k-(Nlines/2))^2)+((i-(Ncolumns/2))^2));
        if (R<totalradius)
            greymatrix(k,i,1)=1.0;
        end
    end
end

figure
imshow(greymatrix)

FFT=fftshift(fft2(greymatrix));
figure
shading flat
mesh((abs(FFT)));
colorbar
```

A Matlab Code Morphology

```
clear all; close all;
se = strel('line',20,100);
Gre = imread('bloodBW.png');
```

```
4
5  %%%Erode image %%%%
6  Greeroded=imerode(Gre,se);
7
8  figure
9  imshow(Greeroded);
10
11 %%%Dilate Image %%%
12 Gredilated=imdilate(Gre,se);
13 figure
14 imshow(Gredilated);
15
16
17 %%%%%Erode second example %%%
18 Gre = imread('images1.jpg');
19
20 Greeroded=imerode(Gre,se);
21 figure
22 imshow(Greeroded);
23
24 %%%%%%%%Dilate it %%%%%%%%%
25 Gredilated=imdilate(Gre,se);
26 figure
27 imshow(Gredilated);
28
29 %%%%%%%%%%%Erode and dilate %%%
30
31 Gredilatederoded=imdilate(Greeroded,se);
32 figure
33 imshow(Gredilatederoded);
34
35 %%%%%%%%%%%Dilate and erode%%%
36
37 Greerodeddilated=imerode(Gredilated,se);
38 figure
39 imshow(Greerodeddilated);
40
41
42 %%Distances from the sea for several points
43
44 Eu = imread('EuropeBW.bmp');
45 A=size(Eu);
```

```
map=zeros(A(1),A(2),101);
map(:,:,1)=Eu;
se = strel('disk',2);

for i=1:256
map(:,:,i+1) = imdilate(map(:,:,i),se);
for j=1:A(1)
    for k=1:A(2)
        if map(j,k,i+1)-map(j,k,i)==1
            mapnew(j,k)=i*256/150;
        end
    end
end
end
figure
imshow(Eu)
figure
imagesc(mapnew)%, colormap gray;
colorbar
```

B Data Analysis

B.1 MATLAB Code kNN Algorithm

```
clear all, close all, clc ;

I = imread('SpainBeach.png') ;

[U,V,d] = size(I) ;
I = double(I) ;

DataBase = [ 161 119 97 1 ;
             171 117 89 1 ;
             164 116 96 1 ;
             255 193 149 1 ;
             210 140 104 1 ;
             255 168 128 1 ;
             255 175 135 1 ;
             141 104 88 1 ;
             110 86 76 1 ;
             91 82 67 1 ;
             79 87 89 2 ;
```

```
19                99  109  98  2  ;
20                98  100  95  2  ;
21                67  76  83  2  ;
22                116  134  112  2  ;
23                94  100  100  2  ;
24                90  82  89  2  ;
25                108  123  116  2  ;
26                92  94  84  2  ;
27                107  107  99  2  ;
28                19  48  66  3  ;
29                26  41  62  3  ;
30                28  92  101  3  ;
31                17  61  72  3  ;
32                27  95  99  3  ;
33                244  246  243  3  ;
34                251  255  255  3  ;
35                20  70  77  3  ;
36                65  94  98  3  ;
37                17  69  82  3  ;
38                141  156  153  3  ;
39                71  118  124  3  ] ;
40
41  NbData = size(DataBase, 1) ;
42
43  Pixels = DataBase(:,1:3) ;
44
45  Classes = DataBase(:,4) ;
46
47  MaskClasses = zeros(U,V) ;
48
49  for u = 1 : U
50      for v = 1 : V
51
52              r = I(u,v,1) ;
53              g = I(u,v,2) ;
54              b = I(u,v,3) ;
55
56              TabRGB = repmat([r g b],NbData,1) ;
57              D2 = (TabRGB - Pixels).^2 ;
58              [valmin,posmin] = min( sum(D2') ) ;
59              MaskClasses(u,v) = Classes(posmin) ;
60
```

```
61
62      end
63  end
64
65  figure,
66  subplot(2,2,1), imshow(uint8(I)) ;
67  title('spain beach - original');
68  subplot(2,2,2), imshow(MaskClasses==1) ;
69  title('spain beach - beach (class 1)');
70  subplot(2,2,3), imshow(MaskClasses==2) ;
71  title('spain beach - vegetation (class 2)');
72  subplot(2,2,4), imshow(MaskClasses==3) ;
73  title('spain beach - water (class 3)');
74
75  set(gcf, 'PaperUnits', 'points');
76  set(gcf, 'PaperPosition', [0 0 900 600]);
77  saveas(gcf,'../images/kNN.png');
```

B.2 MATLAB Code k-Means Algorithm

The Code is based on the K-Means Clustering Example in the MATLAB Documentation and can be found at uk.mathworks.com

```
1  clear all; close all; clc;
2
3
4  he = imread('SpainBeach.png');
5  cform = makecform('srgb2lab');
6  lab_he = applycform(he,cform);
7
8  ab = double(lab_he(:,:,2:3));
9  nrows = size(ab,1);
10 ncols = size(ab,2);
11 ab = reshape(ab,nrows*ncols,2);
12
13 nColors = 3;
14 % repeat the clustering 3 times to avoid local minima
15 [cluster_idx, cluster_center] = kmeans(ab,nColors,'distance','
       sqEuclidean', 'Replicates',3);
16
17 pixel_labels = reshape(cluster_idx,nrows,ncols);
18 %figure, imshow(pixel_labels,[]), title('image labeled by cluster
       index');
```

```
19
20 segmented_images = cell(1,3);
21 rgb_label = repmat(pixel_labels,[1 1 3]);
22
23 for k = 1:nColors
24     color = he;
25     color(rgb_label ~= k) = 0;
26     segmented_images{k} = color;
27 end
28
29
30 figure,
31 subplot(2,2,1), imshow(uint8(he)), title('spain beach - original');
32 subplot(2,2,2), imshow(segmented_images{1}), title('objects in
      cluster 1');
33 subplot(2,2,3), imshow(segmented_images{2}), title('objects in
      cluster 2');
34 subplot(2,2,4), imshow(segmented_images{3}), title('objects in
      cluster 3');
35
36 set(gcf, 'PaperUnits', 'points');
37 set(gcf, 'PaperPosition', [0 0 900 600]);
38 saveas(gcf,'../images/kMeans.png');
```