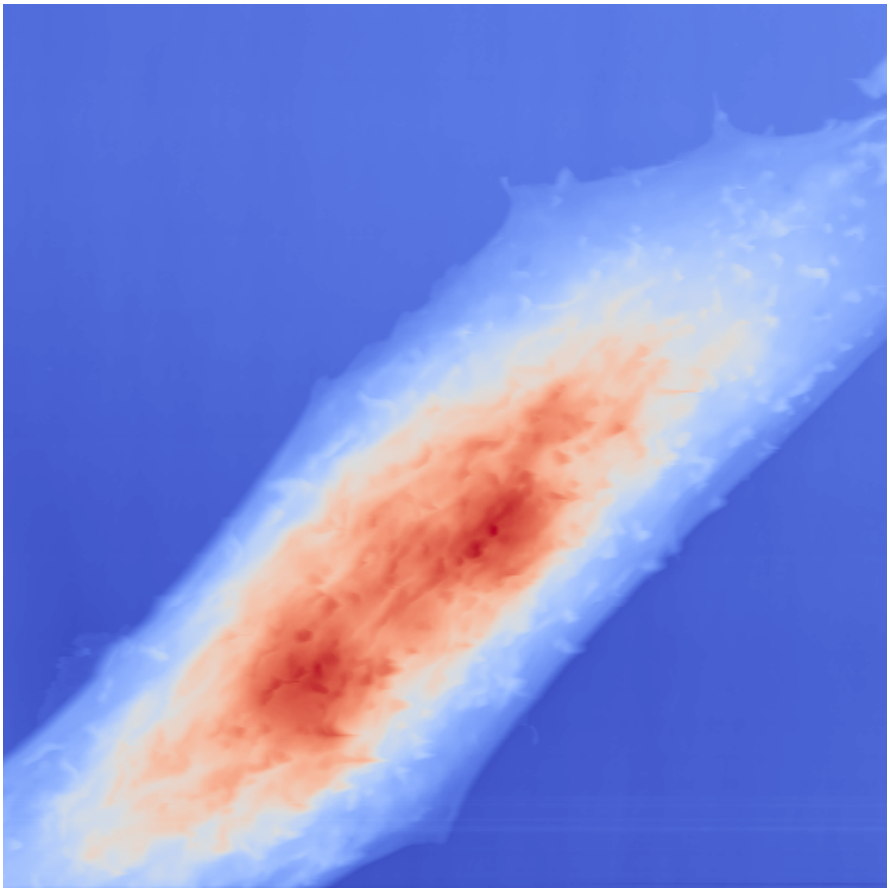
AALBORG UNIVERSITY
STUDENT REPORT

Master's Thesis

Advanced sampling and reconstruction of images in Atomic Force Microscopy



Authors:
Benjamin Højmose Grevenkop-Castenskiold
Jacob Bøgeskov Nøergaard

Supervisor:
Thomas Arildsen

This report is compiled in LaTeX. Figures are made using Inkscape, Python and Matplotlib.

AALBORG UNIVERSITY
STUDENT REPORT

Signal Processing and Acoustics
Aalborg University
http://www.aau.dk

Title:
Advanced sampling and reconstruction of images in Atomic Force Microscopy

Theme:
Master's Thesis

Project Period:
Spring Semester 2019

Project Group:
Group 19gr1071

Participants:
Benjamin H. Grevenkop-Castenskiold
Jacob Bøgeskov Nørgaard

Supervisor:
Thomas Arildsen

Number of Pages: 44

Date of Completion: June 6, 2019

Abstract:

Atomic Force Microscopy (AFM) is a very useful technique to make a topology map of a specimen at a large range of magnifications. However the scanning process can be very time consuming since the probe has to slide across the surface of the sample in order to take the measurements.
The work in this report tries to make this scan time shorter based on cutting edge research in image reconstruction and sample pattern generation.
For reconstruction, there has recently been proposed a way of utilizing the structure of a Neural Network (NN) for image reconstruction without training. This method is called Deep Image Prior (DIP).
A two shot adaptive sample pattern is also proposed where at first a crude scan is performed on the specimen. Then the interesting regions are identified, and scanned with the wanted resolution.
The DIP method for reconstruction was deemed infeasible for AFM due to its reconstruction performance being comparable to that of interpolation, while being much more computationally expensive.
The adaptive sample pattern has promising results for the relevant parts of the image. It shows an average speedup of 10 times for a reconstruction with approximately 44 dB Peak Signal to Noise Ratio (PSNR) for the relevant parts of the image. This speedup can however vary greatly from image to image but the method will ensure that only the most relevant parts of the image is raster scanned.

PREFACE

This project is the Master's Thesis composed by group 19gr1071 of Signal Processing and Acoustics master at Aalborg University.

For citation the report employs IEEE referencing method. If citations are not present by figures or tables, these are made by the authors of the report. Units are indicated according to the SI system. Functions, Python modules and filenames are indicated in `teletype font`.

Aalborg University, June 6, 2019

Benjamin Højmose Grevenkop-Castenskiold
<bgreve14@student.aau.dk>

Jacob Bøgeskov Nørgaard
<jnarga14@student.aau.dk>

TABLE OF CONTENTS

ABBREVIATIONS

AFM	Atomic Force Microscopy
CNN	Convolutional Neural Network
DIP	Deep Image Prior
DNN	Deep Neural Network
MSE	Mean Square Error
NN	Neural Network
PSNR	Peak Signal to Noise Ratio
SSIM	Structural SIMilarity
STM	Scanning Tunneling Microscope
TV	Total Variation

1 INTRODUCTION

Atomic Force Microscopy (AFM) is a microscopy technique that samples the height of a specimen by interacting with it by using a sharp probe. This makes it possible to obtain a nanometer-scale 3D surface, showing features which is not possible to obtain by other microscopy methods. However this is rather time consuming depending on the specimen type and the wanted resolution as the probe has to physically touch the specimen . This process can take several minutes for large images to complete. Recent studies propose ways of decreasing the sample time by either crafting a sample pattern based on a heuristic that the frequency content of a specimen is approximately uniform. This work designs the sample pattern by first raster scanning a small part of the image and then scanning the rest of the image with this designed pattern[1].

Another recent study [2] relies on more advanced reconstruction techniques with simple sample patterns e.g. total variation and interpolation. Here multiple reconstruction techniques are tested with varying scan times.

Another study [3] propose a scan technique where the probe scans until it finds a structure, which it then scans along the edge in different layers. Essentially creating a high quality contour plot.

The work done in [4] does something similar where the boundary of the specimen first is identified by tracing the edge, after which the specimen is raster scanned.

There is also interesting work done on the front of image reconstruction. In [5] the authors show that a Neural Network (NN) does not need to be trained in order to make high quality reconstructions of images.

The rest of this section will mainly be based on [6] unless other is specified.

AFM was developed in 1985 to measure forces smaller than 1 µN between the tip of the AFM probe and the specimen. This was developed on the basis of Scanning Tunneling Microscopes (STMs) which is based on the current drawn between a specimen and the tip of the microscope when a bias voltage is applied. Here the current drawn is used to make the topological image. This has the obvious disadvantage that it only works on conducting samples, giving it a narrow use.

The functioning of AFM varies from STM by measuring the force instead of current. This is done by having the tip on a flexible plate with a reflective surface on top as shown in Figure 1.1 and Figure 1.2. The flex of the cantilever is then recorded with some deflection sensors, thus giving a metric of the amount of force exerted on the tip. This is either used to control the relative movement between the specimen and tip or as a direct information which can be used as topological information. This makes it much more versatile compared to STM.

AFM can be used to magnify from 10^3 to 10^9 times in all dimensions, this makes it possible to study specimen at both the macro and atomic scale. As a comparison, optical microscopes can magnify up to 10^3 times. In addition to being versatile in magnification AFM also works in a variety of different environments like different gases, ambient air, liquids or vacuums. Some live biological specimen require sampling in water which makes AFM ideal.

The normal mode of operation is using a raster scanning pattern as shown in Figure 1.3. Here the probe has to return back to the left side of the workspace without registering any points due to the deflection of the cantilever as shown in Figure 1.4. Recording the data on the returning path would cause a slight shift in the location of the height data

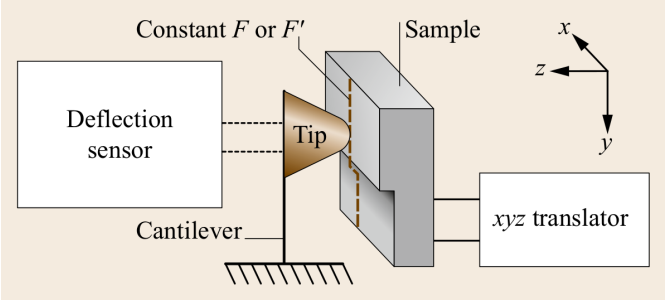


Figure 1.1: Principle of operation for AFM [6]

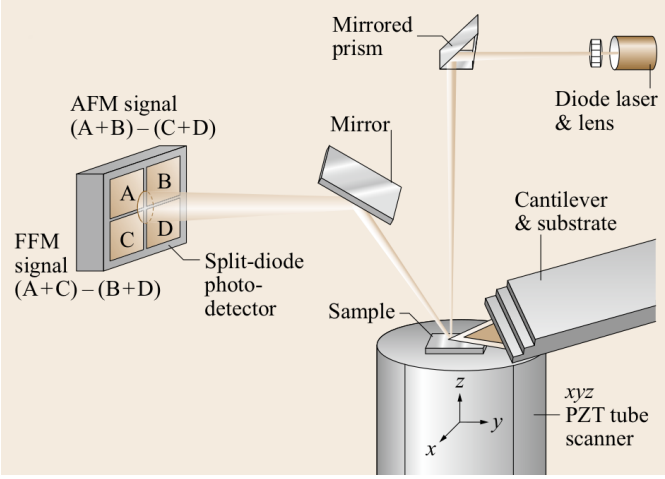


Figure 1.2: Principle of operation for commercial AFM [6]

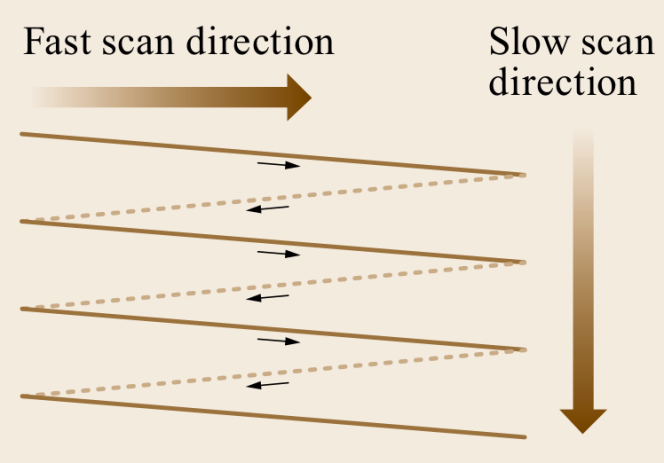


Figure 1.3: Typical scan pattern for AFM. Data is only recorded during the solid scan lines to avoid the tip deflection distortion [6]

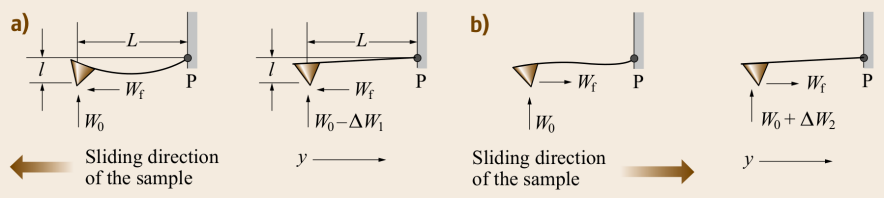


Figure 1.4: Deflection of the cantilever based on the scan direction which causes a height and phase difference in the recorded data [6]

as well as a height difference, making the information troublesome to stitch together. The problem cannot be fixed by the use of calibration because the deflection varies by the physical attributes of the specimen.

The scan time depends on the size of the scanned area. A larger area requires a slower scan speed. The speed is measured in Hz and is a measure of how many times a second the needle moves across the area in the fast scan direction as shown in Figure 1.3. The total time of one scan is then the number of lines in the slow direction divided with the scan frequency in the fast direction. For a large scan length (125 µm), the scan rate is limited to around 0.5 Hz to 2.5 Hz which means for a 512x512 pixel image has a scan time between $\frac{512}{2.5\,\text{Hz}} = 204\,\text{s} \approx 3\,\text{min}$ and $\frac{512}{0.5\,\text{Hz}} = 1024\,\text{s} \approx 17\,\text{min}$. For smaller images, the scan rate is typically around 60 Hz which results in a total scan time of 8.5 s.

AFM has two primary scanning modes of operation. One is a constant contact mode where the needle always is in contact with the specimen, essentially dragging along the surface. This works well for hard surfaces. The other mode is a tapping mode where the needle oscillates close above the surface, tapping the specimen softly. This is shown in Figure 1.5. The advantage of this mode is that it minimizes the effect of friction on soft specimen.

Since the process of scanning with AFM is rather time consuming, it is desired to decrease the scanning time without reducing the quality of the image significantly. In this project the speedup is based on the process of scanning objects that stand out from the background. This makes the desired data appear in cluster at it appears in [3, 4]. However since both these methods require running inside the control loop of the AFM

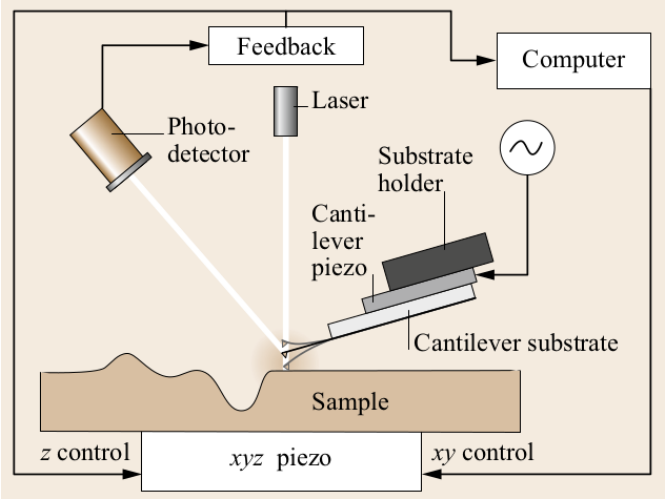


Figure 1.5: AFM tapping mode [6]

microscope, this work will focus on a "two shot" implementation of the same concepts where the specimen first is scanned coarsely, after which the interesting regions are scanned with a higher resolution.

An example of images with localized regions of interest can be the process of scanning cells. A dataset of AFM sampled cells is described in section 1.1 which will be used through the rest of this report. In this project it is desired to be able to adaptively pick out these clusters that are the most interesting and create a pattern design that covers these regions in more detail.

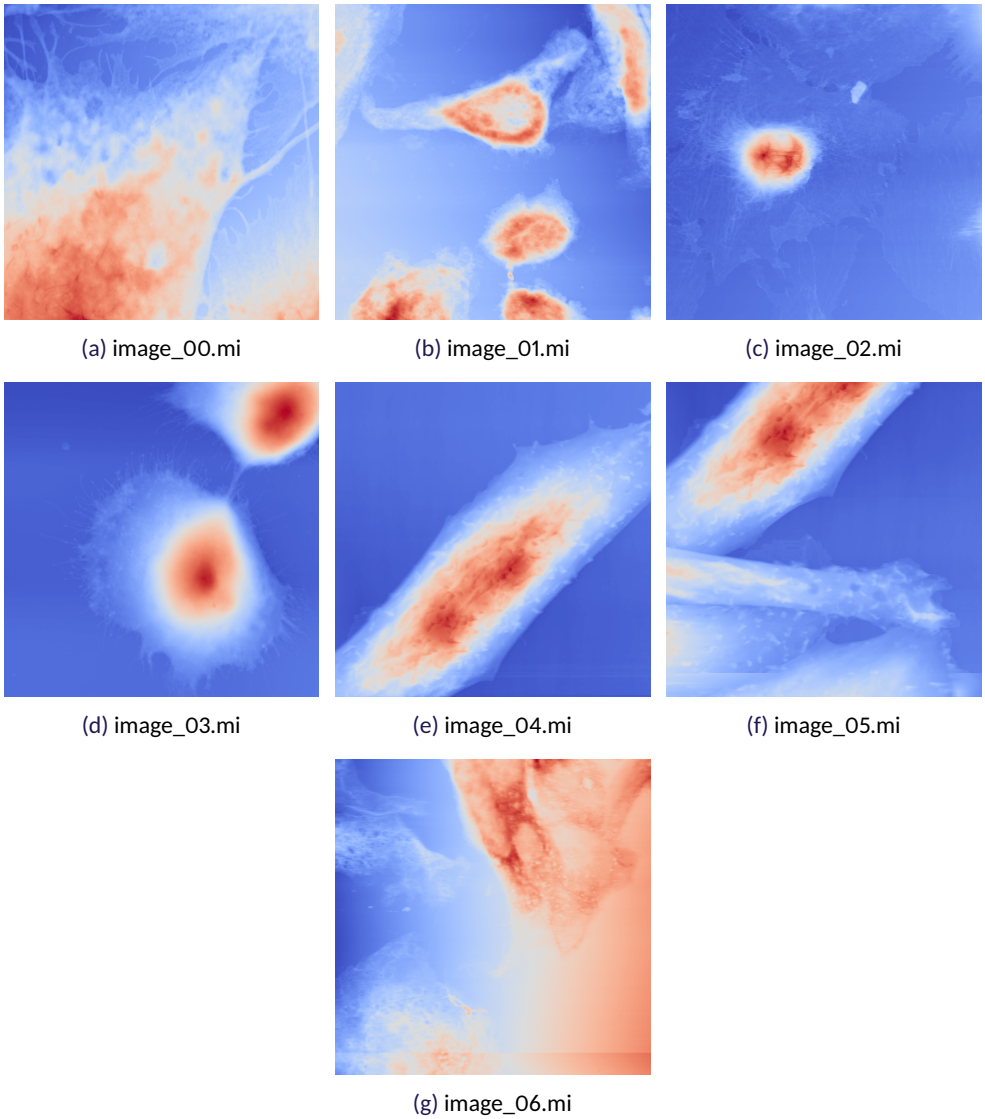
This lead to the description of the dataset.

1.1 DATASET

The used dataset consists of AFM images used in a previous study and can be found in [7]. The images show different cell specimens of Chinese hamster ovary cells and human bladder carcinoma cells. The image format *.mi* is containing the height data in addition to some meta data of the scan. The only data used for this project is the typography tracks of the data in one direction. The python package `Magni` is used to unpack the data and to apply some pre-processing. This package can be found in [8].

The chosen images are shown in Figure 1.6 and shows the type of cell as shown in the following list.

- image_00.mi - Chinese hamster ovary cells
- image_01.mi - Chinese hamster ovary cells
- image_02.mi - human bladder carcinoma cells
- image_03.mi - human bladder carcinoma cells
- image_04.mi - Chinese hamster ovary cells
- image_05.mi - Chinese hamster ovary cells
- image_06.mi - Chinese hamster ovary cells



(a) image_00.mi (b) image_01.mi (c) image_02.mi

(d) image_03.mi (e) image_04.mi (f) image_05.mi

(g) image_06.mi

Figure 1.6: AFM data made available at [7]

1.2 DELIMITATIONS

This project is focusing on creating a solution to speed up the scanning of objects that has locally clustered data such as cells. The work done through this project is thus based on the dataset described in section 1.1 which fulfill this criterion. The main goal of the project is to create a "two shot" sampling pattern that adapts to each specimen individually, creating a high quality image of only the interesting parts of the specimen.

Various reconstruction algorithms are used to recreate the image after sampling with the newly designed sampling pattern. These algorithms will be limited to bilinear interpolation, bicubic interpolation and Deep Image Prior (DIP). DIP is an algorithm that was not tested during the work in [2] but has shown promising results for image reconstruction. It is thus of interest to compare this algorithm to the other reconstruction algorithms. By using some of the same methods for reconstruction as in [2] it is possible to compare the performances.

1.3 PROBLEM STATEMENT

How can the process of AFM scanning be sped up without reducing the quality of the image significantly and without modifying the control loop of the microscope?

2 METHODS

In this chapter the methods used for generation of the iterative sample pattern and the reconstruction of the images using sparse data is presented. First the methods for dividing the image into different segments of interest are introduced, then the reconstruction algorithms that are used and compared are described and at last the metrics used for quality control are defined.

2.1 SAMPLE PATTERN GENERATION

Scanning an object by using AFM is a time consuming process which can take from seconds to hours to perform by using the conventional raster pattern [2]. It is thus favorable to reduce the length of the path and by that reduce the time consumption of the scanning. This can be done by using a sparse sampling pattern. This will give an under sampled representation of the surface which is an effective way to reduce the length of the scanning while the challenge is to retain the quality of the image. In [1] and [2] different pattern designs has been tested, but in this project the scope will be to segment the image into smaller areas of interest and design a new pattern in these areas. This method seems promising since this project is focusing on scanning objects that has data with cluster features as described in section 1.2. This will be done using various methods which are described in this section.

2.1.1 UNDERSAMPLING RATIO

The undersampling ratio is a metric defined in [2] and is used to define how much data that is represented in the sparse representation of the image compared to the total length of a complete raster scan. The ratio is based on the length of the scan since this is approximately proportional to the time it takes to scan the image.

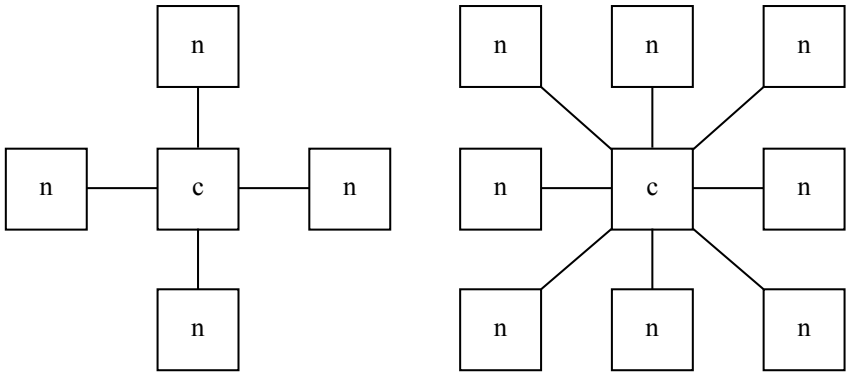
The reference length used, is calculated in Equation 2.1 where w is the horizontal line width in pixels and h is the number of lines. The reason that the product is multiplied by two is because the probe is scanning both back and forth for each line. This creates two images with an offset as described in chapter 1 and only one of these images is used due to the fact that the images are approximately equal but with the said offset.

$$L_{\text{ref}} = 2wh \tag{2.1}$$

With the reference defined the undersampling ratio can be calculated as Equation 2.2 where L is the length of the new sampling pattern in pixels.

$$\delta = \frac{L}{L_{\text{ref}}}. \tag{2.2}$$

Since the length of the path is approximately proportional to the time consumption of the scanning, the undersampling ratio can also be seen as the speedup compared to a full raster scan of the image by multiplying the undersampling ratio with the scanning time of the full raster scan.



(a) 1-connectivity. (b) 2-connectivity.

Figure 2.1: Image segmentation connectivity where ***c*** indicates the center and ***n*** indicates the neighbors.

2.1.2 IMAGE SEGMENTATION

Image segmentation is a way to split the image into different regions. A method of separate the regions is to use pixel connectivity. Pixel connectivity looks at the surrounding neighbors of each pixel in the image. If the values of two neighboring pixels has the same value, or are in the same chosen interval of values, they are connected. Pixel connectivity has two ways of defining neighbor where one gives each pixel four neighbors and the other eight. These two modes are called 1-connectivity and 2-connectivity respectively and is shown in Figure 2.1a and Figure 2.1b. The connectivity number specifies the maximum number of orthogonal hops from the center pixel to consider a pixel as a neighbor as shown in Figure 2.2 [9].

The coordinates of the neighboring pixels to a point is described in Equation 2.3 Equation 2.4 for four and eight neighbors respectively [10].

$$N_4(x, y) = \{(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)\} \tag{2.3}$$

$$N_8(x, y) = N_4(x, y) \cup \{(x + 1, y + 1), (x - 1, y - 1), (x - 1, y + 1), (x - 1, y - 1)\} \tag{2.4}$$

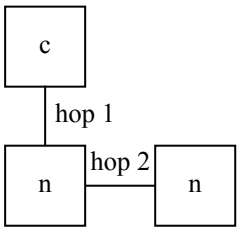


Figure 2.2: Orthogonal hops

Before the segmentation is performed a method to adaptively quantize the values into clusters of values is used.

2.1.3 K–MEANS

The k-means algorithm, also called Lloyd's algorithm, is a method to cluster samples into different groups. This is done by taking a number of N samples and divide them

into k clusters C. These clusters are described by the mean, μ_j, of the samples in each cluster. These means are also called the centroids of the clusters and are not necessarily a part of the original set of samples [11]. The algorithm is described in pseudocode in Algorithm 1.

Algorithm 1 k-means algorithm

1. **Input:** data, number of clusters
2. Create k initial centroids randomly
3. While (old centroid - new centroid) > threshold
   (a) Assign samples to closest centroid
   (b) Calculate new centroids from the mean of the samples of the assigned to each cluster
4. **Output:** Cluster centroids

From Algorithm 1 it is seen that the algorithm is initialized by creating k initial centroids randomly. After that the algorithm assigns each sample to the nearest centroid creating k clusters. From each cluster of samples a new centroid is calculated by taking the mean of said cluster. These two last steps are repeated until the difference between the old and the new centroid is equal to zero or below some threshold, which means that the centroid does not move significantly. The algorithm then outputs the centroid of each cluster and the k-means of the data is available [12] [11].

The algorithm minimizes the sum-of-squares problem shown in Equation 2.5.

$$\sum_{\mathrm{i}=0}^{N} \min_{\mu_j \in C} (\|x_i - \mu_j\|^2) \tag{2.5}$$

An example of the k-means algorithm in one dimension is shown in Figure 2.3. This example show how three means calculated on linear data with a value from 0 to 1.

This can be used to quantize the pixel values of an image into N regions which has the minimum Mean Square Error (MSE).

2.1.4 MAX–POOLING

Max pooling is a sample based discretization process used to reduce the number of parameters present in the image. This can be used to quantize an image into regions represented by some values represented in a quantization codebook. These values can for example be determined by the k-means algorithm described in section 2.1.3. The concept of max pooling is visualized in Figure 2.4 where it is shown that a 4×4 pixel image is separated into regions of 2×2 pixels. The image is then mapped into a 2×2 image by taking the maximum value of each region which is then used to represent the whole region but in a downsampled image.

Since it can be desired to have an image of same resolution as the original image, the max pooled image can then be upsampled as shown in Figure 2.4 and each region is now represented by the maximum value of each region but has the same resolution as the original image.

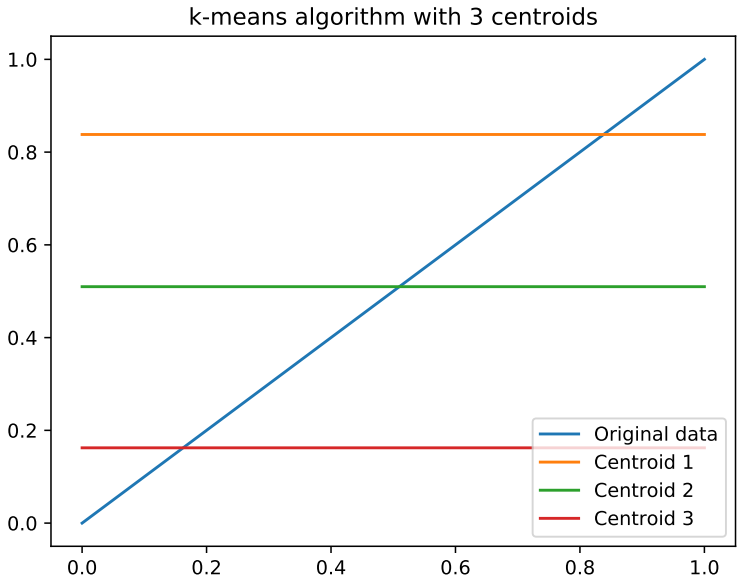


Figure 2.3: Example of the k-means algorithm on linear data with 3 centroids.

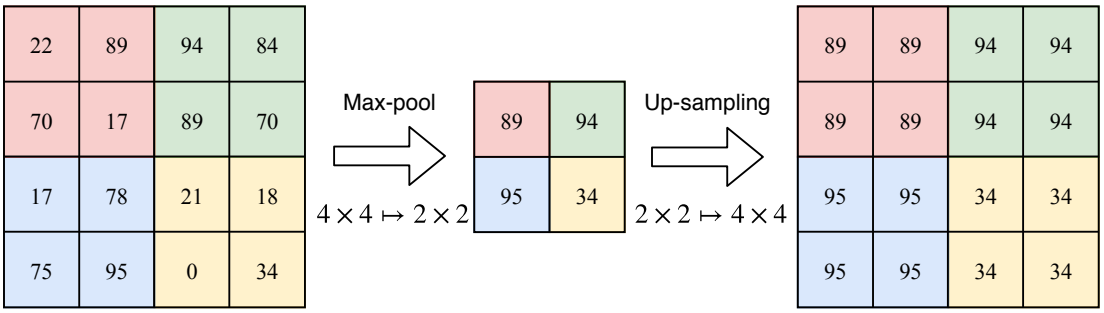
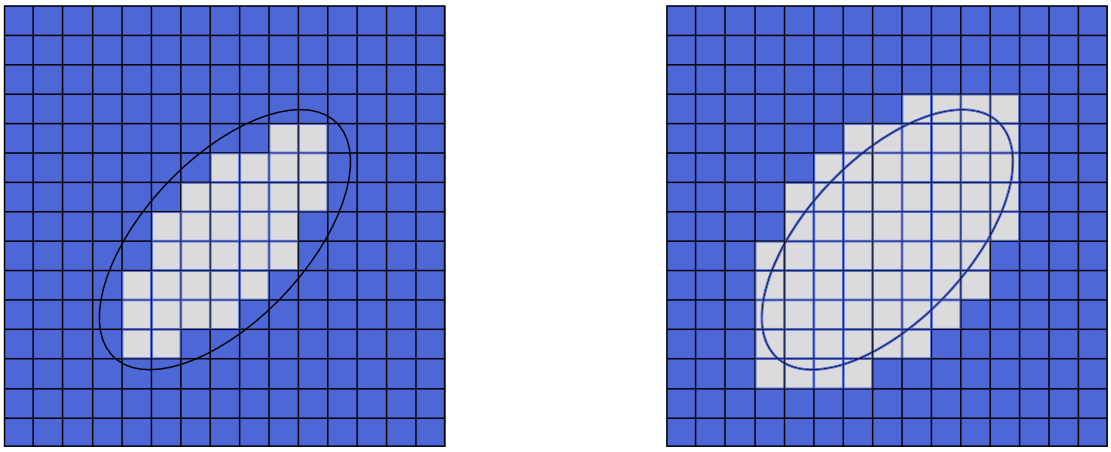


Figure 2.4: Concept of max pooling

Since the scope of this project is to focus on local regions of an image, it is preferable select data outside the region which is acheived with max pooling rather than min pooling. As shown in Figure 2.5 max pooling is overlapping the region of interest which is intended since the borders of the region is not represented in min pooling as shown in Figure 2.5a.



(a) Min pooling.

(b) Max pooling.

Figure 2.5: Example of min pooling and max pooling.

Hereby the methods used for pattern generation is concluded and the reconstruction algorithms can be explored.

2.2 RECONSTRUCTION ALGORITHMS

This section contains the theory of different reconstruction algorithms used for reconstructing a image from a sparse input. This section will describe how the DIP method can be used to to reconstruct images from a sparse input and its limitations. Next a description of different types of interpolation that also can be used to reconstruct an image from a sparse representation.

2.2.1 DEEP IMAGE PRIOR

In image processing there has recently been studies that show that it is not necessary to train a Deep Neural Network (DNN) in order to make image restoration. Instead the image is reconstructed by exploiting that NNs are better at representing the structures of natural images than random noise [5]. The rest of this section is based on [5] unless other is stated.

In regular image restoration a minimization problem of the type of Equation 2.6 is often used. Here $E(x; x_0)$ is a problem dependent data term and x_0 is the noisy image which it is desirable to apply the reconstruction algorithm on. Lastly a regularization term, $R(x)$, is added which makes the resulting images look natural.

$$x^* = \min_x E(x; x_0) + R(x) \tag{2.6}$$

The choice of the data term $E(x; x_0)$ is application dependent and therefore has a specific form for the given problem. However, the regularization term is what makes the

images look natural which is no easy task. This is not tied to a specific application and therefore this is where much of the previous research has been done. One of the simpler examples is Total Variation (TV) which makes images have uniform color regions. The work done in [13] provides solutions to the problems denoising, inpainting and deblurring using the TV regularization term in different optimization problems. However TV was originally proposed in [14] for denoising images. The TV is the sum of the variation in the image. This is done by taking the sum of the difference between neighboring pixels in both directions. In [15] the discrete version of TV is defined as Equation 2.7.

$$TV(x) = \sum \sqrt{(x_{\mathrm{i+1,j}} - x_{\mathrm{i,j}})^2 + (x_{\mathrm{i,j+1}} - x_{\mathrm{i,j}})^2}. \tag{2.7}$$

This regularization method favors that different image regions has the same height which is a good assumption for images.

However, the proposed minimization problem in [5] is shown in Equation 2.8 where the regularization term is removed and the optimization variable is changed to the parameters of a NN.

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E(f_\theta(z); x_0) \tag{2.8}$$

$$x^* = f_{\theta^*}(z) \tag{2.9}$$

This optimization problem finds the network parameters that minimizes the energy metric between the output of the NN and the known pixels from the image. The result of the optimization problem is then the result of the random input with the optimized network parameters. These network parameters are then in turn used to reconstruct the image as shown in Equation 2.8

The input for the NN z is initialized with random noise at the start of the optimization problem. It is important to note that it is not the input vector that is optimized over but the weights of the network as in traditional NNs. A small amount of noise on the input can be beneficial in order to regularize the solution further.

This method have been proved useful for a wide range of image restoration purposes. Examples given includes denoising, super resolution and inpainting. However only inpainting is described in this report as the difference to denoising and super resolution is a simple change to the energy function and network parameters and not the overall structure of the problem.

INPAINTING

One of the shown uses for this method is inpainting. Here a binary mask is given for a image where some pixels of the image is missing. An example of this is shown in Figure 2.6 where the information missing from the overlaying text is reconstructed.

The energy for the inpainting optimization problem is calculated by

$$E(x; x_0) = ||(x - x_0) \odot m||^2 \tag{2.10}$$

where x_0 is given as the data of the known pixel values corresponding to the mask m which has the same dimensions as x and contains zeros at the places of the unwanted pixels and ones at the wanted pixels. Here $\odot$ denotes the Hadamard product which is the element wise multiplication of two matrices of equal size. This is the MSE of the relevant pixels of the image.

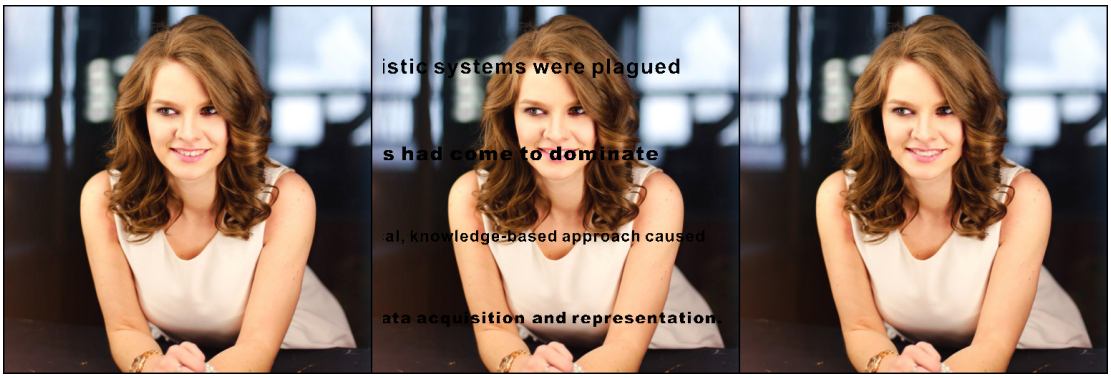
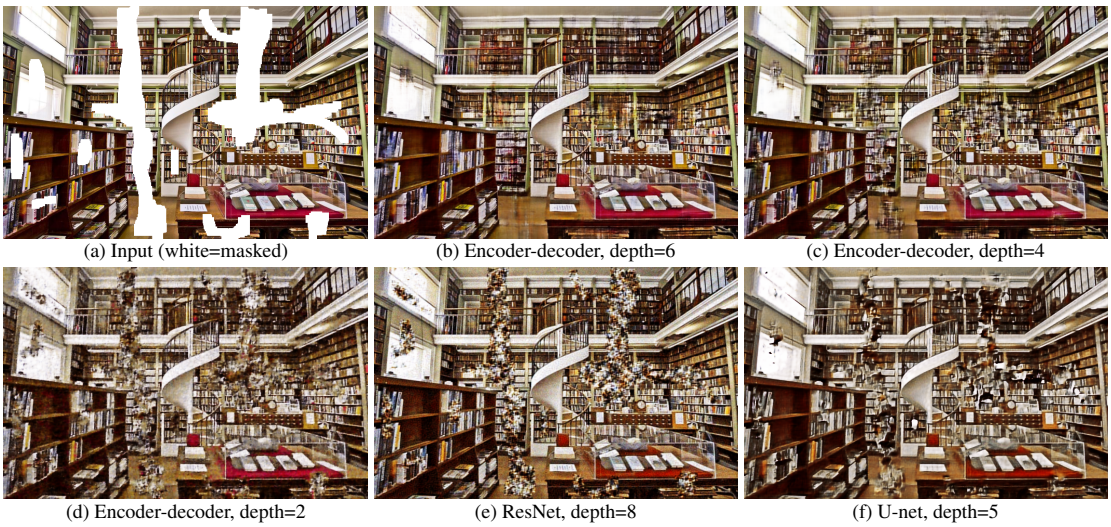


Figure 2.6: Image showing inpainting of missing information. On the left the original image is shown. The middle image is the image with added text and on the right is the reconstructed image with a Peak Signal to Noise Ratio (PSNR) of 41.76 dB. [5]



(a) Input (white=masked) (b) Encoder-decoder, depth=6 (c) Encoder-decoder, depth=4

(d) Encoder-decoder, depth=2 (e) ResNet, depth=8 (f) U-net, depth=5

Figure 2.7: Image showing inpainting of missing information for different network structures. [5]

This example shows inpainting for relatively small areas but the method is not limited to this. Large hole inpainting is shown in Figure 2.7 where large parts of the image is reconstructed without prior knowledge about the nature of the image. Here, the artifacts produced by different network structures is also shown. The result is not perfect, but is rather good considering that the network is not trained.

This leads into the different network structures used.

NEURAL NETWORK CONFIGURATIONS

The proposed method can be utilized on any NN structure, but the authors of [16] found a Convolutional Neural Network (CNN) auto encoder with some skip connections most successful for most purposes. All the used network structures are provided in the supplementary material at [16]. However the default parameters is shown in Table 2.1.

These network parameters is then inserted into the network structure shown in Figure 2.8 where the auto-encoder network is defined.

In general, NNs covers a range of function approximators which fits a simple structure

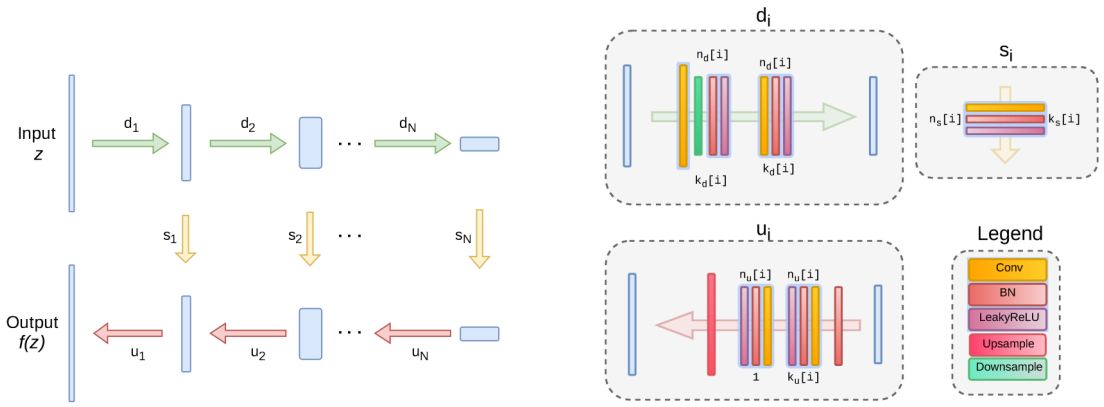


Figure 2.8: Default architecture used for DIP. This structure is a auto-encoder network where $n_u[i], n_d[i], n_s[i]$ corresponds to the number of filters for the upsampling, downsampling and skip connections at depth i respectively. The values $k_u[i], k_d[i], k_s[i]$ is the corresponding kernel sizes [16]

Table 2.1: Default network parameters from [16]

Parameter	**Variable name and value**
Random input	$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
Upsampling and downsampling: number of filters	$n_u = n_d = [128, 128, 128, 128, 128]$
Upsampling and downsampling: kernel size	$k_u = k_d = [3, 3, 3, 3, 3]$
Skip-connections: number of filters	$n_s = [4, 4, 4, 4, 4]$
Skip-connections: kernel size	$k_s = [1, 1, 1, 1, 1]$
Regularization noise	$\sigma_p = \frac{1}{30}$
Number of iterations	num_iter $= 2000$
Learning rate	LR $= 0.01$
Upsampling type	upsampling $=$ bilinear

to a unknown or complex function by iterating over the relationship between the input and outputs of the function. A famous example of this in action includes the recent results by Google where they used machine learning for mastering the game of Go without human knowledge [17].

A bunch of different NN architecture types can be utilized for different purposes. For images there is CNN which excel at problems which is shift invariant. Examples of this includes detection of objects in images, since the label does not change based on where in the image the object appears. This is due to CNNs containing a operation of weighing values closely spaced together with a kernel which can propagate through the network. This learned kernel can highlight simple features like edges which in later layers can be combined into shapes which can be combined into objects and so on.

An auto-encoder network is a NN which narrows at the middle which means that it has to represent the data at the input as sparsely as possible, extracting only the essential information, for then to recreate the information again on the other side. This makes it possible to try to generate new ways of compressing data, or make encoding algorithms for noisy transmission types.

In as a contrary to DIP, NNs often requires very large amounts of data in order for the network structure to generalize to the underlying function. This makes the DIP result more impressing. Since DIP is the usage of a NN without the necessary training, a more in depth description of NN is omitted. For further information about NNs see [11, 18]

The optimization problem shown in Equation 2.8 is still solved using the same methods as is used traditionally with NNs which is back propagation. Back propagation is a method of distributing the error on the output of the NN after applying an input, backwards to all the nodes in the network. This is done layer by layer iteratively. This individual error on all the nodes can then be used to update the values of all the nodes by moving towards a minimum.

LIMITATIONS

The DIP method is not capable of restoring large holes with complex features due to not being trained. An example of this can be seen in Figure 2.9. This is the main difference between a trained NN and DIP. A trained network can in theory learn that humans have faces and then fill in the image accordingly where DIP merely fills in information that connects nicely to the rest of the image.

2.2.2 INTERPOLATION

In this subsection, a short description of interpolation as a image reconstruction algorithm is descriped. This is done due to the work done in [2] is showing that interpolation is a fast and simple reconstruction algorithm which makes it suitable as a benchmark to compare reconstruction results with.

Interpolation is the term of estimating the value between two samples. Different kinds of interpolation can be used for different cases. The simplest variation is linear interpolation where a straight line is drawn between two points as shown in Figure 2.10 and the intermediate values simply are evaluated using Equation 2.11.

$$f(b) = f(a) + \frac{b-a}{c-a}\left(f(c) - f(a)\right) \tag{2.11}$$

(a) Image of person with face removed with a mask.

(b) Restored image after 5000 iterations with DIP.

Figure 2.9: Image showing inpainting of complex features.

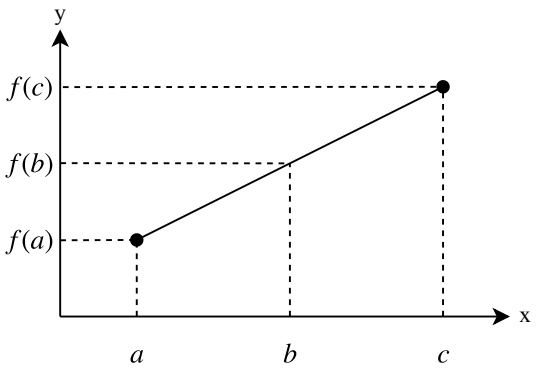


Figure 2.10: Linear interpolation between two points.

Another type of interpolation is cubic-spline interpolation. Instead of connecting the data points using a straight line they are connected using a third degree polynomial of the form of Equation 2.12 where α and β are coefficient that change for each new interval. This is done by using 4 points instead of 2. The added points are used to estimate the differential at the two most centric points.

$$f(x) = \alpha_3 x^3 + \alpha_2 x^2 + \alpha_1 x + \beta \tag{2.12}$$

An comparison of the two types of interpolation and the true data is shown in Figure 2.11. It is seen that in cubic-spline interpolation follows the curve of the true signal while the linear interpolation connects the data points with a straight line. This makes the cubic-spline interpolation represent the original signal better than the simple linear interpolation in cases where the signals are smooth in nature. In cases where sharp edges are precent, the cubic interpolation has a tendency to overshoot the value, as is shown between point 9 and 10 in the figure.

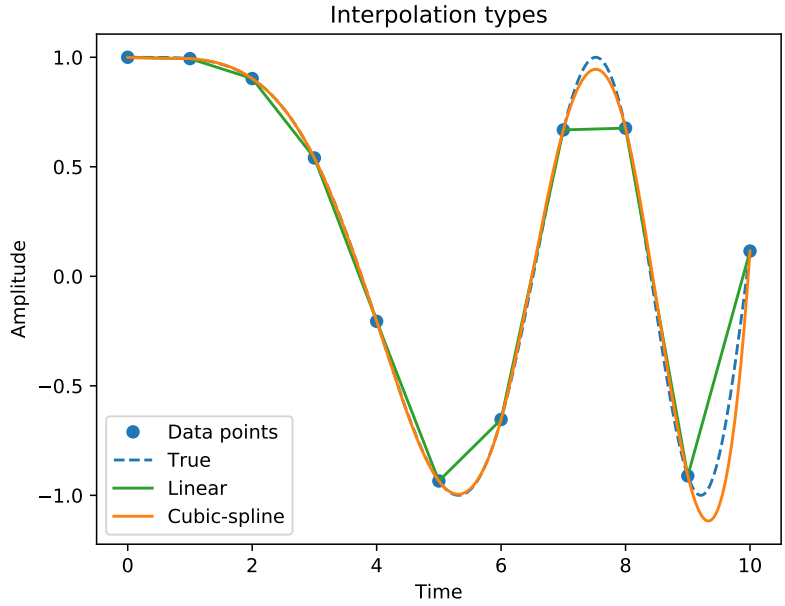


Figure 2.11: Comparison of different interpolation types.

Interpolation in 2D is called bilinar for linear interpolation and bicubic for cubic interpolation. The interpolation is calculated using the python package `scipy` which contains different utilitys for interpolation like `scipy.interpolate.griddata`. This utility can both be used for bilinear and bicubic interpolation.

2.3 QUALITY CONTROL METRICS

As stated in the problem statement in section 1.3 it is of interest to compare the quality of the reconstructed images to the ground truth images. This is done by using different methods such as Structural SIMilarity (SSIM) and PSNR and these methods are described in this section.

2.3.1 STRUCTURAL SIMILARITY INDEX

SSIM is based on [19] and the following theory will not cite further to the article.

SSIM is a method to evaluate the quality in a perceptual manner. A block diagram of SSIM is shown in Figure 2.12 where the x and y signal is assumed to be non-negative image signals.

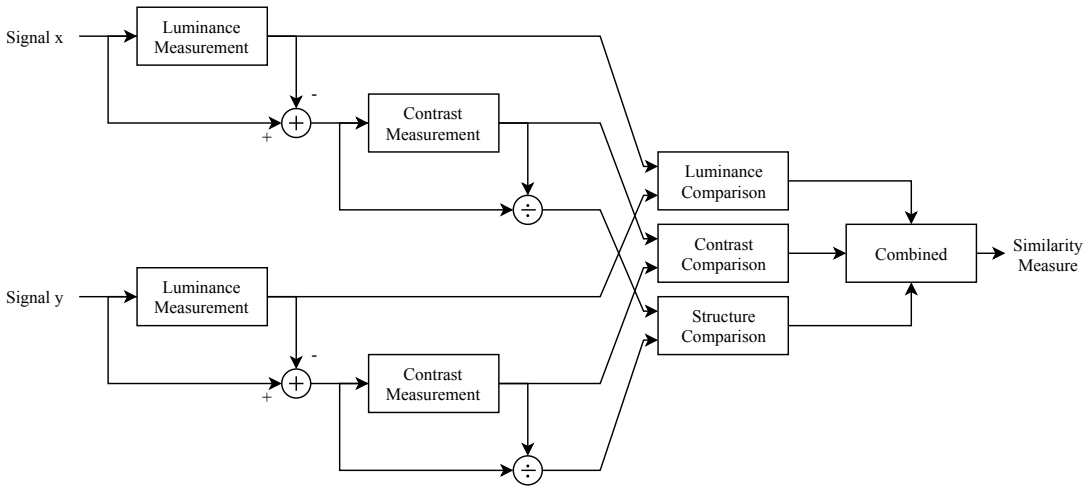


Figure 2.12: Block diagram of SSIM [19].

The perceptual similarity in an image can be calculated by comparing three parameters: luminance, contrast and structure. To calculate the luminance similarity, the mean intensity is calculated in Equation 2.13 assuming discrete signals.

$$\mu_x = \frac{1}{N}\sum_{\mathrm{i}=1}^{N} x_i \tag{2.13}$$

The luminance comparison function, $l(x, y)$, is thus a function of μ_x and μ_y. As shown in Figure 2.12 the mean is then subtracted from the signal to and thus remove the mean from the signal. To estimate the contrast in the image, the unbiased standard deviation is calculated. This is defined by Equation 2.14 and the contrast comparison $c(x, y)$ is the comparison of the standard deviation of the two signals, σ_x and σ_y.

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{\mathrm{i}=1}^{N}(x_i - \mu_x)} \tag{2.14}$$

At last the signal is normalized by dividing the signal with its own standard deviation and the structural comparison function, $s(x, y)$, is a function of this normalized signal $(x - \mu_x)/\sigma_x$ and $(y - \mu_y)/\sigma_y$. The final similarity measure is a function of these three relatively independent functions which gives Equation 2.15.

$$S(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{2.15}$$

The three comparison functions, $l(x, y)$, $c(x, y)$ and $s(x, y)$ as well as the combination function $f()$ is now to be defined.

The luminance is defined as Equation 2.16 where C_1 is a small constant to avoid instability if the numerator becomes close to zero and is defined as Equation 2.17 with $K_1 \ll 1$ is a small constant and L is the dynamic range of the pixel values.

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.16}$$

$$C_1 = (K_1 L)^2 \tag{2.17}$$

Very similar to Equation 2.16, the contrast comparison is calculated as Equation 2.18 where $C2 = (K_2 L)^2$ and $K_2 << 1$.

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.18}$$

$$C_2 = (K_2 L)^2 \tag{2.19}$$

The comparison of the structure is performed after the luminance subtraction and contrast normalization. The correlation between the normalized signals are an effective way to quantify the similarity of the structure of the two signals. The correlation between the two normalized signal are equal to the correlation coefficient between the original signals, x and y thus the structural comparison function is defined as Equation 2.20.

$$s(x,y) = \frac{\mu_{\mathrm{xy}} + C_3}{\mu_x\mu_y + C3} \tag{2.20}$$

As in the luminance and the contrast comparison a small constant, C_3, is introduced to avoid dividing by zero and the covariance coefficient defined as Equation 2.21.

$$\sigma_{\mathrm{xy}} = \frac{1}{N-1}\sum_{\mathrm{i}=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{2.21}$$

Finally the SSIM of the two images can be calculated by combining the three comparisons from Equation 2.16, Equation 2.18 and Equation 2.20 into Equation 2.22.

$$\mathrm{SSIM}(x,y) = [l(x,y)]^\alpha \cdot [c(x,y)]^\beta \cdot [s(x,y)]^\gamma \tag{2.22}$$

Where

$$\alpha > 0, \beta > 0, \gamma > 0 \tag{2.23}$$

α, β and γ are coefficients to weight the different parameters. To simplify the expression in Equation 2.22, the parameters α, β and γ are set to one and $C_3 = C_2/2$ which gives the final result of the SSIM shown in Equation 2.24.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{\mathrm{xy}} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.24}$$

To obtain the overall quality measure for the whole image which is calculated by taking the mean SSIM which is easily done as shown in Equation 2.25 where M is the number of local windows in the image.

$$\mathrm{MSSIM}(X,Y) = \frac{1}{M}\sum_{\mathrm{j}=1}^{M}\mathrm{SSIM}(x_j, y_j) \tag{2.25}$$

In [19] the results of using SSIM versus using more traditional comparison methods as MSE can give a better insight in the perception of the image instead of only looking at the numerical differences in the images.

2.3.2 MEAN SQUARE ERROR

The MSE is often used to compare images. The MSE takes the mean of the squared error between the two images as shown in Equation 2.26. [20]

$$\mathrm{MSE} = \frac{1}{MN} \sum_{\mathrm{i}=0}^{M-1} \sum_{\mathrm{j}=0}^{N-1} \left[I(i,j) - K(i,j) \right]^2 \tag{2.26}$$

Where

M and N are the dimensions of the image

I and K are the two images (2.27)

The PSNR is easily described from the MSE.

2.3.3 PEAK SIGNAL TO NOISE RATIO

The PSNR is defined by Equation 2.28 through Equation 2.30 and is the ratio between maximum power of a signal and the noise. The PSNR is expressed in dB.

$$\mathrm{PSNR} = 10 \cdot \log_{10} \left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}} \right) \tag{2.28}$$

$$= 20 \cdot \log_{10} \left(\frac{\mathrm{MAX}_I}{\sqrt{\mathrm{MSE}}} \right) \tag{2.29}$$

$$= 20 \cdot \log_{10}(\mathrm{MAX}_I) - 10 \cdot \log_{10}(\mathrm{MSE}) \tag{2.30}$$

Where

MAX_I is the maximum value of the image pixels

MSE is the mean square error

The PSNR is often used to measure the quality compression with lossy compression codecs which makes it suitable in this context. [20]

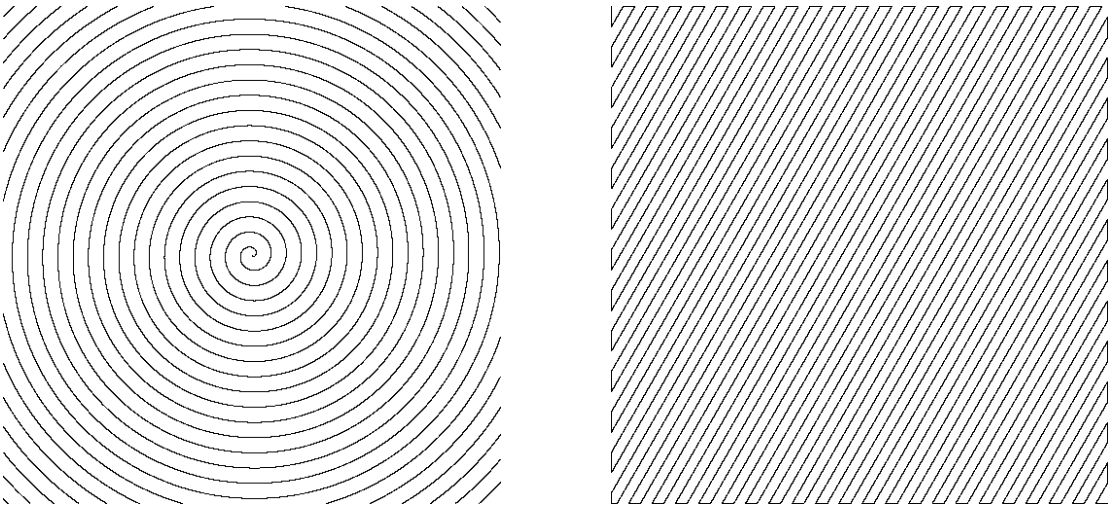
With the methods and the theory behind them described, the design phase can begin. The design will be described in the following chapter.

3 DESIGN

In this section, the different methods from chapter 2 are applied in order to solve the stated problem. The methods will be combined in order to design the adaptive sampling pattern and the reconstruction algorithms are implemented and some tests are made to evaluate the performance. For this, a baseline is introduced to have a foundation, with non-adaptive sampling patterns and simple reconstruction, to compare later results to.

3.1 BASELINE

In order to compare results for the rest of the report, a baseline is created. This is done by recreating some of the results from [2]. The chosen algorithms are bicubic and bilinear interpolation on a spiral and rotated line raster pattern. The sampling patterns are shown in Figure 3.1a and Figure 3.1b. These are created using the python package `Magni` [8]. The `Magni` module is created by the authors of [2].



(a) Spiral pattern. (b) Rotated line raster.

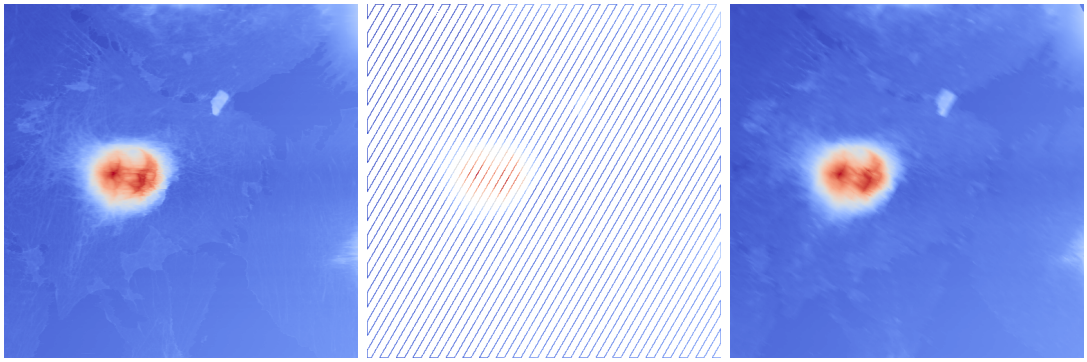
Figure 3.1: The two chosen static patterns for reconstruction.

These two sample patterns are used as shown in Figure 3.2 where the original image in Figure 3.2a is sampled such that only the samples that lie on the path is returned. This is shown in Figure 3.2b. This is then reconstructed with bilinear interpolation using the `scipy.interpolate.griddata` function. The results are shown in Figure 3.1b where some artifacts are present from the low undersampling ratio.

This example is then repeated for the graphs shown in Figure 3.3 and Figure 3.4a.

These graphs show how well the two interpolation techniques reconstruct the image when given a specific sample pattern and undersampling ratio. The graphs are created by evaluating the PSNR and SSIM at 100 different undersampling ratios varying from 0.005 to 0.3. The bilinear interpolation method seems to be best for extremely low undersampling ratios while for undersampling ratios of above 0.125 starts to see a small performance increase with bicubic interpolation.

Since the project is aiming to speed up the process of scanning AFM images, it is desired to compare the time it takes for the reconstruction algorithm to reconstruct



(a) The original image.

(b) Example of sampled image with rotated line pattern applied.

(c) Reconstructed image using bilinear interpolation.

Figure 3.2: Reconstruction example with undersampling ratio of 0.01 and an angle of 30 degrees.

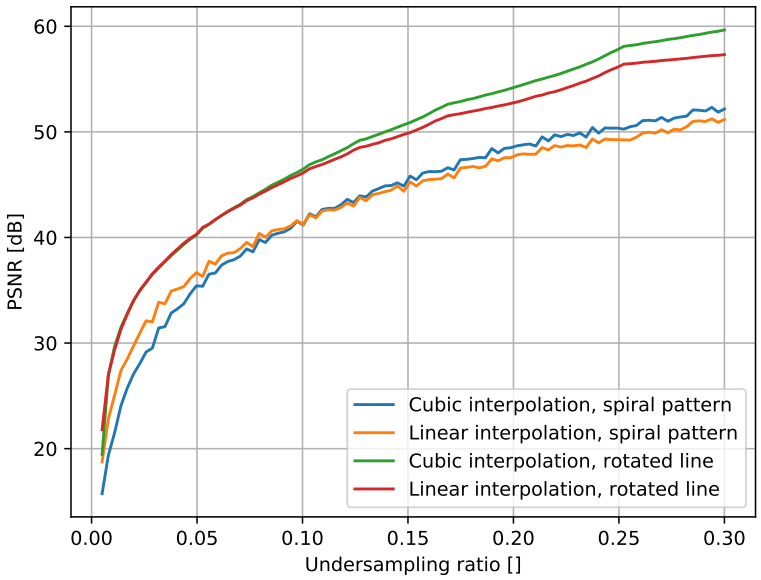
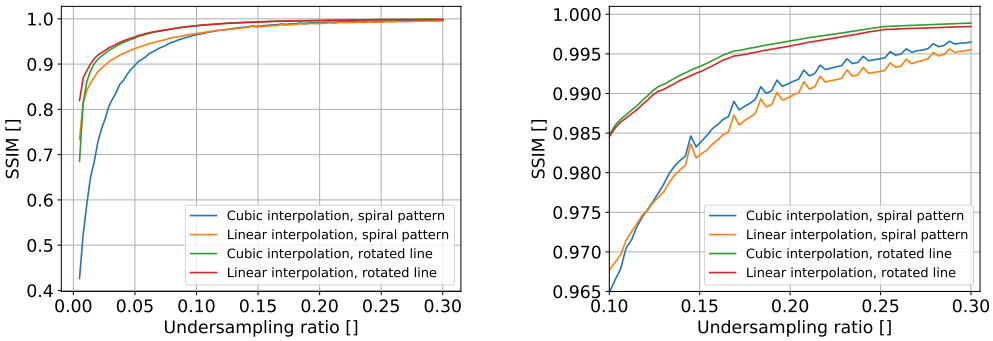


Figure 3.3: PSNR of different sample patterns at different undersampling ratios



(a) SSIM of different sample patterns at different undersampling ratios.

(b) Zoomed in version of Figure 3.4a for better visibility.

Figure 3.4

the image. A comparison of the time is shown in Figure 3.5 with a maximum time at approximately 2 seconds which is equal to the results in [2].

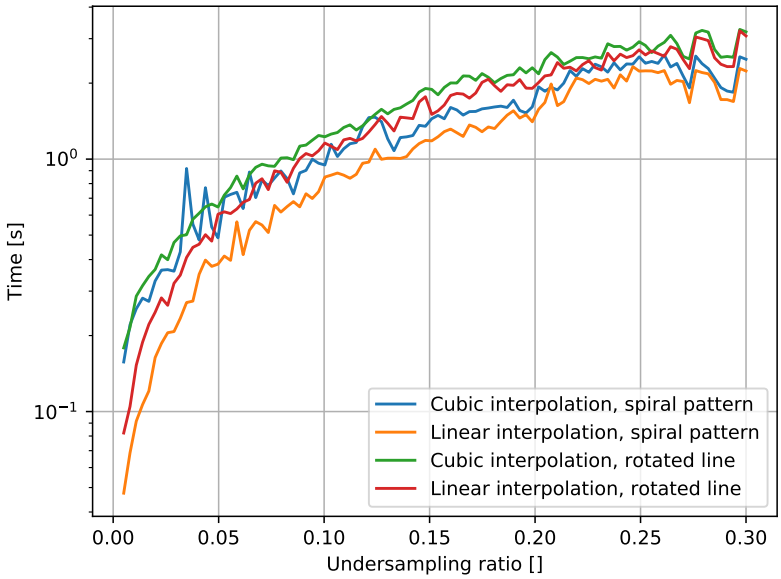


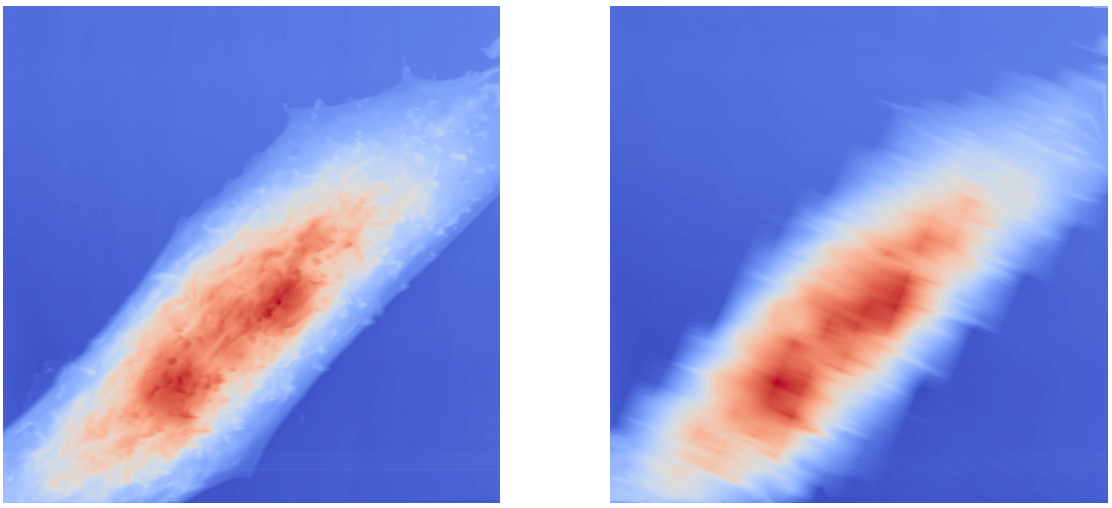
Figure 3.5: Plot showing the time it takes to reconstruct the image using interpolation

The higher PSNR and SSIM of the rotated line could be caused by the fact that the spiral pattern counts the distance traveled outside the square of the image while the raster pattern only samples within the image as shown in Figure 3.1. The reason that the spiral pattern is scanning outside of the square of the image to include the corners of the square. The distance traveled outside will thus not give any new data but the distance traveled is still included since the the probe would still travel the distance. These results are the baseline the rest of the results in this work.

3.2 ADAPTIVE PATTERN GENERATION

In this section the method described in section 2.1 will be combined and used to generate an adaptive pattern design. The goal of the design is to divide the image into three regions of interest based on the values of each sample. This will be done by quantizing to three values where the highest values represent the most interesting part of the image. The outcome will be a pattern with a varying resolution over the image based on the regions of interest.

To get a sense of the content of the image, a very low resolution scan is performed on the image. In this case the undersampling ratio is 0.02. This sparse representation is then interpolated to create an image to work further with. The original image compared to the coarse representation is shown in Figure 3.6.



(a) Original image.

(b) Interpolation after first scan with under sampling ratio of 0.02.

Figure 3.6: The original and the coarse representation of the image.

When the coarse representation of the image is achieved, the quantization is performed. To calculate the quantization values, the k-means algorithm is used as described in section 2.1.3. A plot of all values of the image sorted and the calculated means is shown in Figure 3.7.

With the quantization values of the image calculated, all the values are quantized as shown in Code snippet 3.1.

```
if sample > Cluster 1
    sample = 1
else if sample > Cluster 2
    sample = 0.5
else
    sample = 0
```

Code snippet 3.1: Image quantization pseudo code

This results in the quantized image shown in Figure 3.8a. The three layers are labeled as [Background, Mid, Top].

As shown in Figure 3.8a the two regions has rough edges and small holes. To overcome this, max pooling is used. Max pooling is used to down sample the image into segments

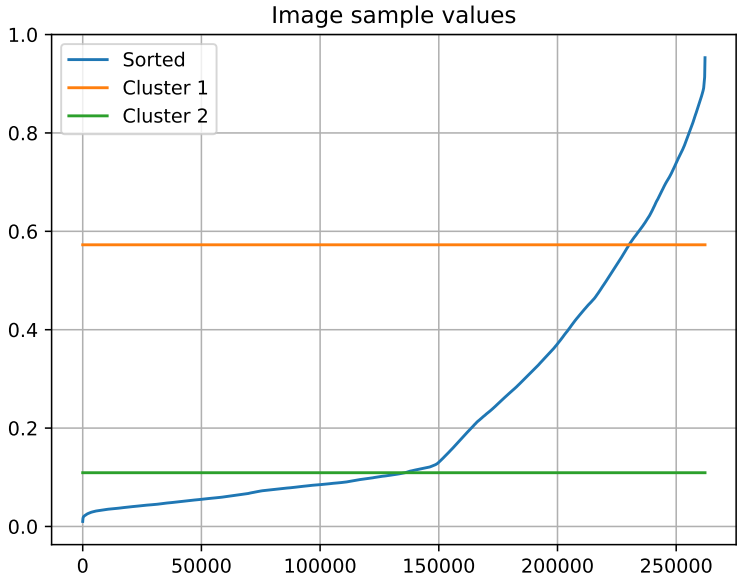
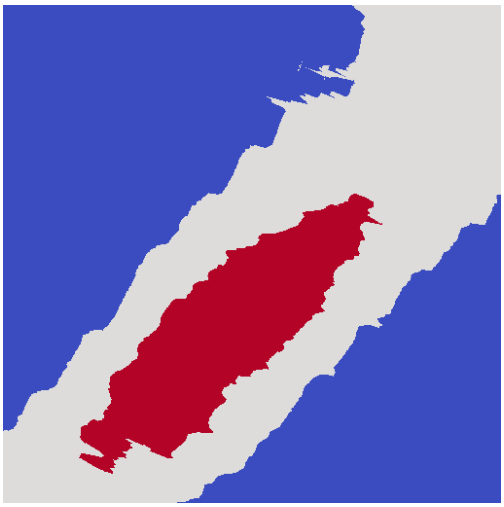
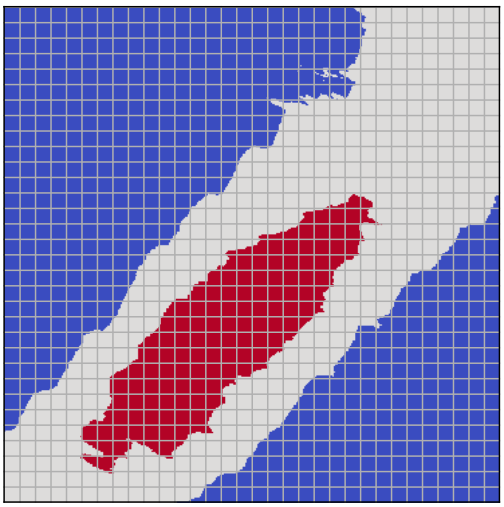


Figure 3.7: K-means cluster values.

of 16 pixels as shown in Figure 3.8b, and all pixels of each segment is set to the maximum value of the given segment.



(a) Quantized version of the crudely reconstructed image



(b) Quantized image with overlayed grid for max pooling

This gives an image without small holes and a clean transition from one region to another as shown in Figure 3.9. When using max pooling there will also be an overlap from the most important region to the lesser important region and makes sure that nothing from the most important region is left out. The result of the max pooling is shown in Figure 3.9.

When the regions of interest are clearly defined as in Figure 3.9, the pattern design can begin.

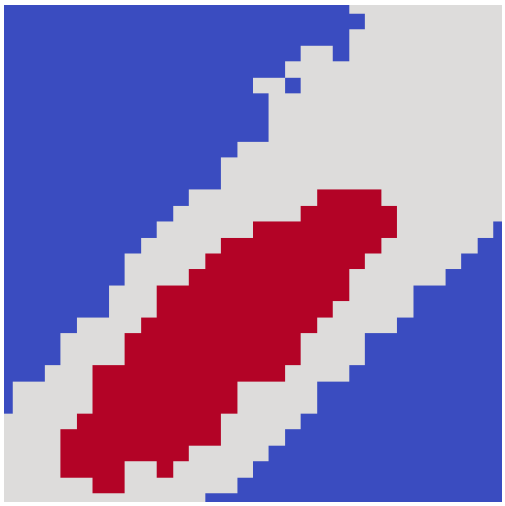


Figure 3.9: Image showing the maxpooled image.

3.2.1 PATTERN DESIGN

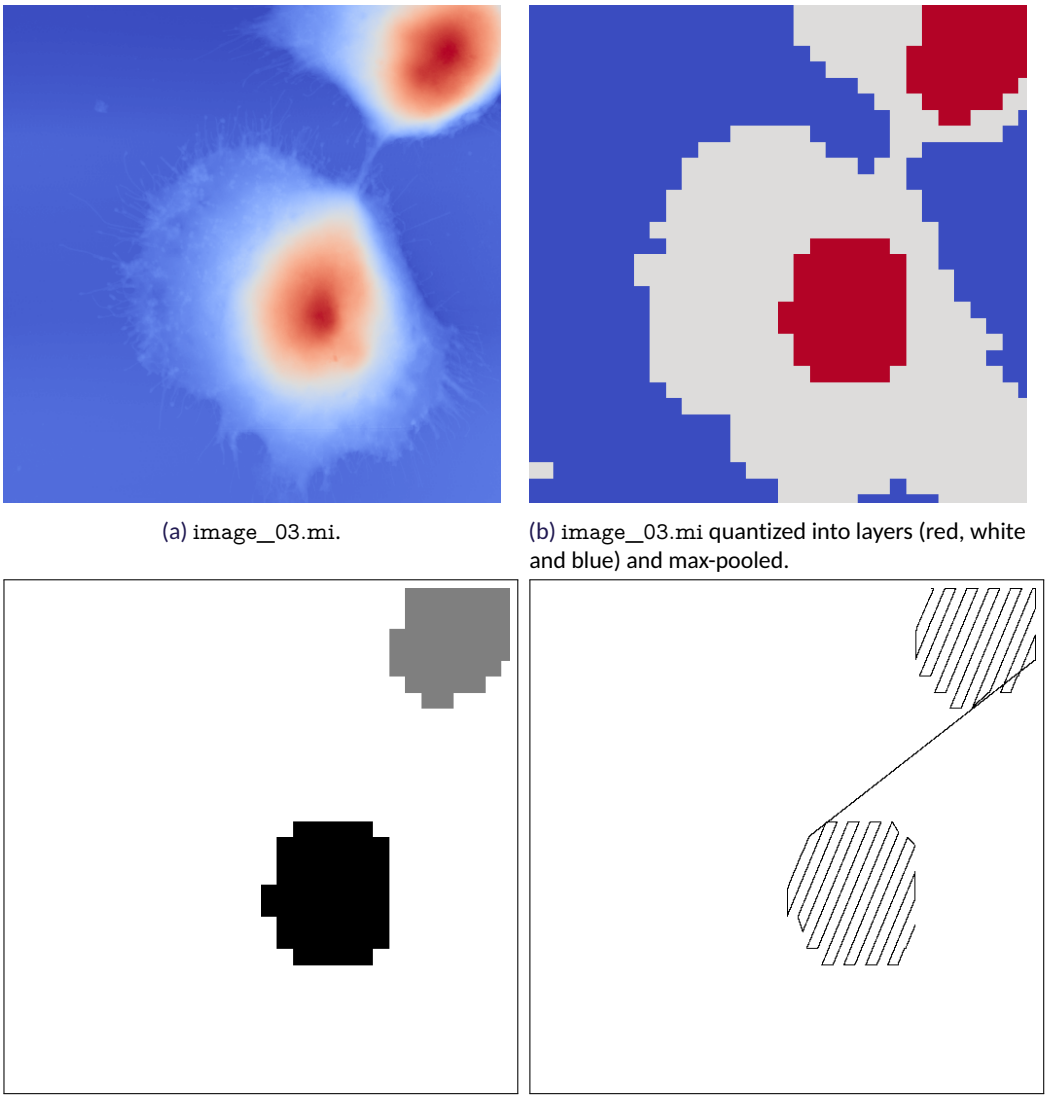
After the interesting layers of the specimen has been identified, each region in the layers are identified. This is done by using the `skimage.measure.label` function which separates regions in a image by clustering all the pixels in the same layer together which are directly next to each other. This is done to the picture in Figure 3.10b where the red region is divided into sub regions as shown in Figure 3.10c. These regions are independently sampled with the same undersampling ratio, and then connected by a direct path as shown in Figure 3.10d. This segmented sampling of the image is repeated for all regions in all layers except for the background layer as this has already been sampled in the first pass.

The sample pattern is applied to the regions by following the coordinates for a generated sample pattern for the whole image. Each coordinate is checked to be inside or outside relevant region. Each time the pattern reenters the region, a straight line is added connecting the reentry point to the exit point. The pseudo code for this algorithm is described in Code snippet 3.2.

```
for current_region in regions:
        path = generate path of wanted length
        for point in path
            if point in current_region
                if last_point not in current_region
                    Connect previous point to this point
                else
                    Add point to new path
```

Code snippet 3.2: Pseudo code for the application of different resolution sample patterns to the different regions

in Code snippet 3.3, the code for sampling the new image is shown.



(a) image__03.mi.

(b) image__03.mi quantized into layers (red, white and blue) and max-pooled.

(c) The two regions in the red layer are separated.

(d) Each region is sampled independently and connected.

Figure 3.10: Separation and sampling of different regions in a specimen.

```
# Add the first raster scan to path
new_coords = path_coordinates.tolist()
# Ignore the first level as this already is sampled for the crude reconstruction
for height in levels[1:]:
        # separate the levels
    regions = sk.measure.label(board == height)
    connected = True

    # Skip first element in order to ignore the surrounding area
    for region_idx in np.unique(regions)[1:]:
        for cord in p[height]:
            if regions[cord[0], cord[1]] == region_idx:
                if connected == True or len(new_coords) == 0:
                    new_coords.append(cord)
                elif len(new_coords) > 0:
                    fromdat = new_coords[-1]
                    todat = cord
                    res = connect(fromdat, todat, (img_h,img_w))
                    new_coords.extend(res)
                    connected = True
            else:
                connected = False
```

Code snippet 3.3: Applying different density sample patterns to the different layers and regions of the image

This code is applied to the initially reconstructed image from Figure 3.6b in order to create the sample pattern in Figure 3.11a. This sampling pattern can then be applied to Figure 3.6a in order to simulate a new scanning with the adaptive pattern design.

The undersampling ratios and angles of the underlying sample patterns are chosen as shown in Table 3.1. The angles are increasing as to minimize the amount of pixels that are scanned multiple times and thus achieve more data from the same scan length. The undersample ratios are chosen rather low to be able to see a visual difference between the adaptive sampling pattern and the raster pattern of same length as a visual example for this report.

Table 3.1: Choice of constants for the adaptive sample pattern.

	Background	Mid	Top
Undersampling ratio [·]	0.02	0.05	0.1
Raster Angle [degrees]	22.5	82.5	142.5

In order to evaluate the performance of this sample pattern, the length of the adaptive sample pattern is calculated. The equivalent undersampling ratio is estimated by

$$\text{usr} \approx \frac{n_{\text{path}}}{2 \cdot w \cdot h}, \tag{3.1}$$

where n_{path} is the number of pixels in the path and w and h is the width and the height of the image in pixels respectively. This approximation will return a slightly smaller undersampling ratio than the true value. However this is determined negligible as each layers sample pattern is at an angle to the previous layer. This reduces the overlap of the different paths. This estimation is used to generate a raster pattern of equal length as shown in Figure 3.11b. Both the sample patterns are then used to reconstruct the image with bilinear interpolation. This is shown in Figure 3.11c and Figure 3.11d. The images both have artifacts from the low sample ratio, but the image reconstructed with the uniform raster pattern has captured the shape of the cell well but is more blurry when it comes to the cell nuclei. The adaptive sample pattern however has some sharp

Table 3.2: Comparison of PSNR and SSIM for the reconstruction based on the proposed adaptive sampling pattern and a raster scan of the same length.

	Adaptive sample pattern	Raster scan
PSNR	42.01 dB	44.20 dB
SSIM	0.980	0.982
undersampling ratio	0.08	0.08

Table 3.3: Comparison of PSNR and SSIM for the different regions of importance as shown in Figure 3.12.

	Adaptive sample pattern	Raster scan
PSNR Figure 3.12a, mid	42.62 dB	44.53 dB
PSNR Figure 3.12b, top	54.75 dB	48.83 dB
SSIM Figure 3.12a, mid	0.984	0.984
SSIM Figure 3.12b, top	0.999	0.995

edges around the cell but has captured the cell nuclei at a higher accuracy. The PSNR and SSIM performance is slightly higher for the raster scan of the same length as is shown in Table 3.2.

These results seem to indicate that the current solution is lackluster to raster scanning at an equivalent undersampling ratio, but since much of the error of the presented sampling pattern belongs to artifacts in less important regions of the image, another way of comparing the reconstructions is needed. This is done by comparing the reconstructed image to the original image in the identified most interesting regions separately. The chosen regions are the whole cell and the cell nuclei for itself. This is shown in Figure 3.12 where the original image is quantized into layers of interest. These layers are then used as a mask for both the adaptive sample pattern and the raster pattern. The results show that the adaptive sample pattern is comparable for the mid region but better on the top layer as is shown in Table 3.3. This method also makes it possible to choose a resolution for the different regions beforehand without having to sample the whole image.

3.3 DEEP IMAGE PRIOR

In this section, the adaptation of the algorithm from section 2.2.1 is shown in order to evaluate the method for use in a AFM context.

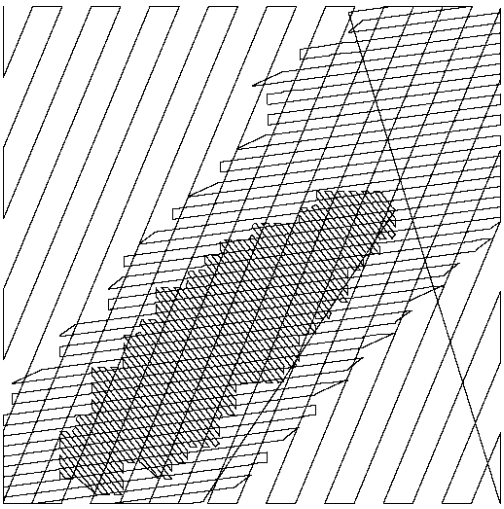
The used energy function for the optimization problem shown in Equation 2.8 is the same as for hole inpainting as the problems are compatible. This is shown by first defining the energy function as

$$E(x; x_0) = ||(x - x_0) \odot m||^2, \tag{3.2}$$

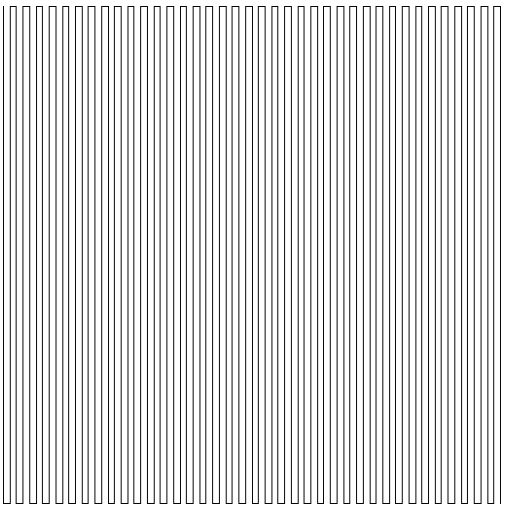
where $\odot$ denotes the Hadamard product which is the element wise multiplication of two matrices of equal dimensions. Additionally the variables x, x_0 and m denotes the reconstructed image, the original image and the mask respectively. Rewriting the equation to

$$E(x; x_0) = ||(x - x_0) \odot m||^2 = ||x \odot m - x_0 \odot m||^2 \tag{3.3}$$

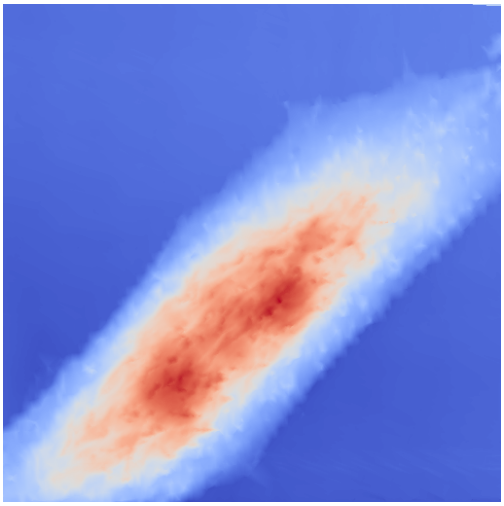
shows that this is merely the MSE of the two images where the mask m is applied. This means that no information outside the chosen mask is used in the optimization problem, making it usable as a AFM reconstruction algorithm.



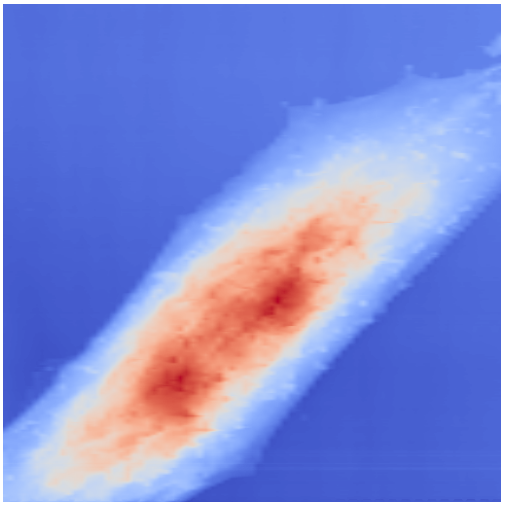
(a) Resulting sample pattern from applying Code snippet 3.3 to Figure 3.6a with an estimated undersample ratio of 0.075.



(b) Raster pattern of same length as Figure 3.11a.

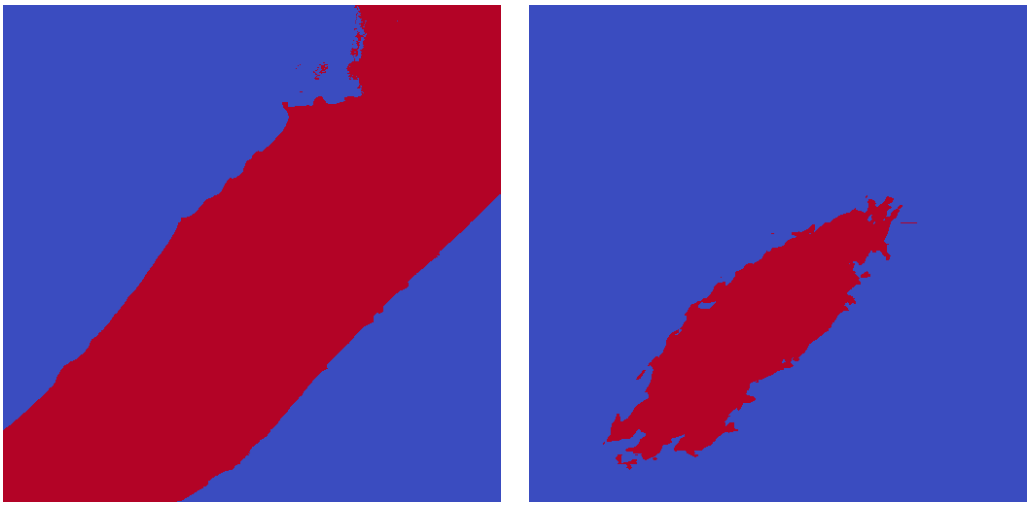


(c) Reconstructed image using adaptive pattern from Figure 3.11a.



(d) Reconstructed image using raster pattern from Figure 3.11b.

Figure 3.11: Sample patterns of same length and their respective reconstruction using linear interpolation.



(a) Mask containing the whole cell. (b) Mask covering only the nuclei of the cell.

Figure 3.12: Masks for more localized comparison.

Since code for the work done in [5] is readily available as python code using `pytorch` at [21], only small adjustments are needed to adapt it to use with AFM.

Here the example `inpainting.ipynb` is chosen as a starting point. This file is stripped down until only the essentials are left. The essentials are described further through the rest of this section.

First the data needs to be loaded in order to pass it to DIP. This is done using the `Magni` package. This is done by running the following snippet.

```
1 img = magni.afm.io.read_mi_file("image_00.mi").get_buffer('Topography')[0].data
```

Here the `Topography` buffer is loaded as this is the height data of the sample. Then the first of the two tracks is loaded and the data is returned. A small wrapper doing this is made in the file `imageloader.py`. This small utility also has the option to load `.png` files and also implements a simple re-sampler.

As the implementation of DIP is already made, the default network configuration is made as following.

```
1 net = skip(input_depth,
2            img.shape[0],
             num_channels_down = [128, 128, 128, 128, 128],
4            num_channels_up   = [128, 128, 128, 128, 128],
             num_channels_skip = [4, 4, 4, 4, 4],
6            filter_size_down = 3, filter_size_up = 3, filter_skip_size=1,
             upsample_mode='bilinear',
8            downsample_mode='avg',
             need_sigmoid=True, need_bias=True, pad=pad).type(dtype)
```

The `skip` function is defined by the authors of [5]. This is passed to a closure function together with the data. The closure function is used to do the forward backward pass of the NN. This function is then passed to the optimizer which runs this for `num_iter` iterations, fitting the network to the image. It is also in the closure function intermediate results can be evaluated and saved. In the following code snippet a stripped down version of the closure function is shown. The lines 7-10 are the only ones that are totally necessary as this is where the data is sent through the network structure, the loss is

calculated and the back propagation is applied. On line 9, the same energy function as in Equation 3.3 is applied. Here both the output of the neural network and the original image has the mask applied as to only compare the masked regions. Based on this masked loss, the optimizer then takes the network parameters a step in the direction that minimizes this.

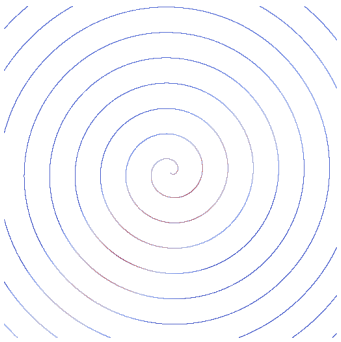
```
1  def closure():
2    global i
     net_input = net_input_saved
4    if reg_noise_std > 0:
         net_input = net_input_saved + (noise.normal_() * reg_noise_std)
6
     out = net(net_input)
8
     total_loss = mse(out * mask_var, img_var * mask_var)
10   total_loss.backward()

12   if  PLOT and i % show_every == 0:
       print ("Iteration {}    Loss {}".format(i, total_loss.item()), end='\r')
14     out_np = torch_to_np(out)
       plt.imsave(f'{i}.png', out_np)
16
     i += 1
18
     return total_loss
20
   optimize(OPTIMIZER, p, closure, LR, num_iter)
```

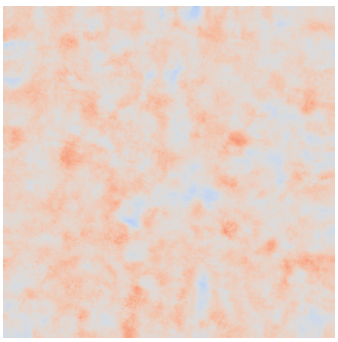
As an example, a reconstruction is made using a very low undersampling ratio of 0.02. This is run for 10000 iterations with otherwise default network parameters. Some of the intermediate results are shown in Figure 3.13 and the true and masked PSNR is shown in Figure 3.14 where the masked PSNR is comparing the reconstructed image only within in sampled region while the true PSNR compare the reconstructed image to the whole original image. The images are saved at 50 iteration intervals and shows the intermediate outputs for the NN. All the reconstructed images can also be found in the attached material under `attachments/deep-image-prior/recon_example/0.02/`.

On Figure 3.14 it can be seen that the NN settles into a minimum until just after 8000 iterations where it decides to jump out of the minimum. This means that basing the results on iterations alone comes with the risk of having a good solution which is not the one after the set number of iterations. On Figure 3.13 the path from a random noise output to the finished reconstruction.

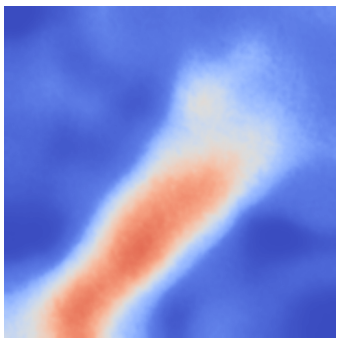
With the design of the adaptive sampling pattern done and DIP configured, the results can be retrieved. The results will be a comparison of different types of reconstruction algorithm and comparing the adaptive sampling pattern with a raster scanning pattern of equal length.



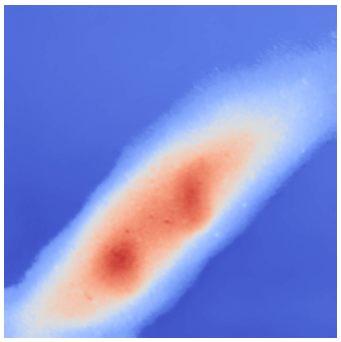
(a) sampled image with a undersampling ratio of 0.02.



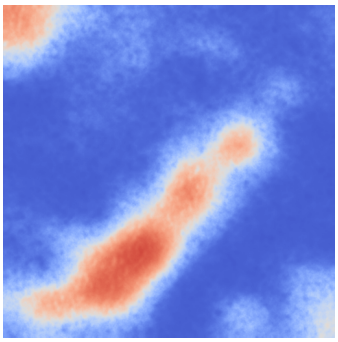
(b) Output of the NN at 0 iterations.



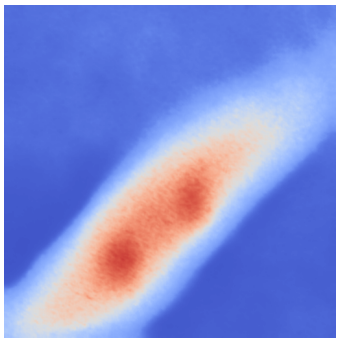
(c) Output of the NN at 100 iterations.



(d) Output of the NN at 5000 iterations.



(e) Output of the NN at 8450 iterations and after the drop in PSNR.



(f) Output of the NN at 10000 iterations.

Figure 3.13: The output of the DIP network at different iterations.

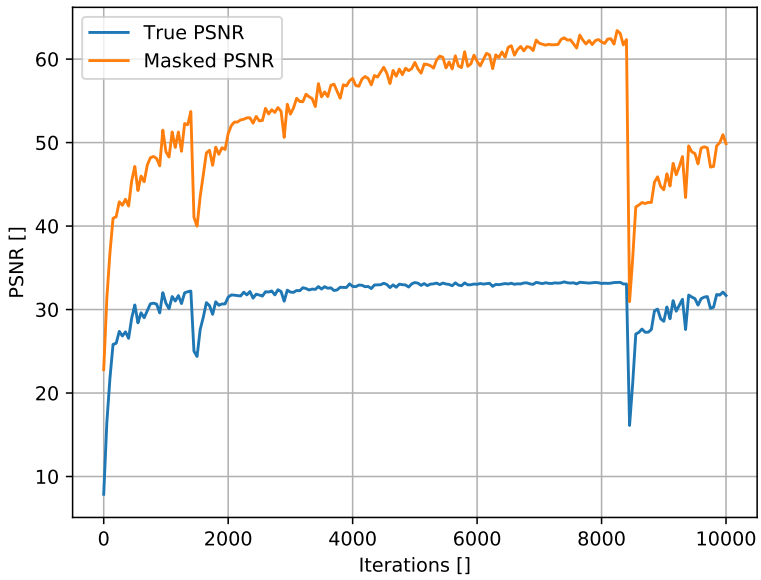


Figure 3.14: Plot showing the real and the masked PSNR for 10000 iterations of DIP reconstruction.

4 RESULTS

In this section the performance of DIP and the adaptive sample pattern generation is presented in a AFM context. The results will show the performance of DIP compared to bilinear and bicubic interpolation both in terms of quality and time consumption. The results will also reflect the performance of the adaptive sampling pattern compared to a raster pattern of equal length. The comparison will be done in the whole image but also in the selected clusters to see the effect of the quality for the different regions.

4.1 DEEP IMAGE PRIOR

In order to evaluate whether DIP is worth pursuing for the purpose of reconstruction in the context of AFM, it is compared against the baseline in section 3.1.

On Figure 4.1 the DIP method is compared to the baseline. Here the reconstruction performance is comparable to interpolation for small undersampling ratios. However as the undersampling ratio increases, the performance decreases. This could be due to the stopping criteria being hard coded to 2000 iterations. For the SSIM the performance is also comparable at lower undersampling ratios as shown in Figure 4.2 but show a similar decrease in performance when the undersampling ratio increase like for the PSNR.

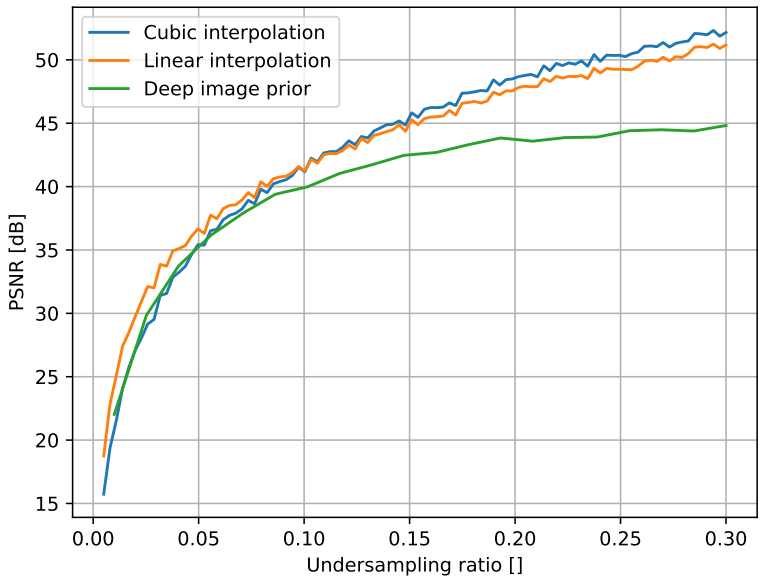


Figure 4.1: Comparison between the baseline and DIP for PSNR for a spiral pattern.

On Figure 4.3 the runtime for different undersampling ratios is shown. The constant time consumption is due to the set stopping criterion of 2000 iterations and the time will only increase by adding iterations. This makes the DIP method unfit for AFM, and thus this project, as in order to make the reconstruction acceptable, the reconstruction time would likely be larger than the time it would take to raster scan the whole image. There is also little need for this as bilinear and bicubic interpolation seems to have the same or better performance at much less computational cost. It is therefore decided that

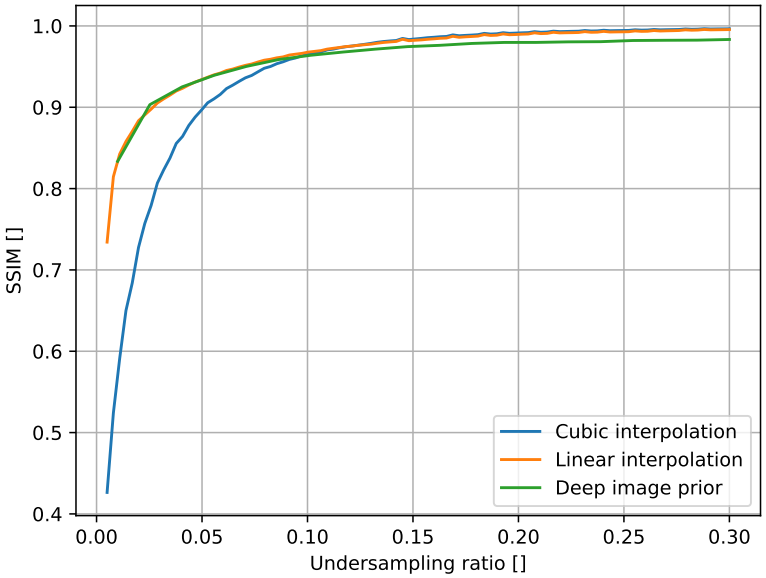


Figure 4.2: Comparison between the baseline and DIP for SSIM for a spiral pattern.

the DIP method is not pursued or tested further in this work.

4.2 ADAPTIVE SAMPLE PATTERN

Here the performance of the adaptive sample pattern design is evaluated. This is done using the implementation in the file `barchart_generation_for_all_images.py` in folder `attachments/code/adaptive_sample_pattern/`

All the intermediate images for all shown reconstructions are found at `attachments/adaptive-pattern-results`. The files and folders are named according to the three undersampling ratios in increasing order from the initial scan up to the top layer. The different mesurements are noted as [all, mid, top] and covers the whole image, from the mid layer and up and lastly only the top layer. Results from more configurations than shown in this report are also available in the attachments.

The results show two different combinations of undersampling ratios, namely $[0.02, 0.05, 0.10]$ and $[0.02, 0.10, 0.20]$. For these two setting the PSNR and the SSIM is compared by subtracting the results of the raster pattern of equal length from the adaptive sampling pattern. The difference between the two is shown in Figure 4.4 and Figure 4.5 for the PSNR and Figure 4.7 and Figure 4.7 for the SSIM. These graphs show that for all and mid the performance is mostly better when using the raster scan but at the top layer the adaptive pattern performs best.

A comparison of the reconstructed image when using the adaptive pattern and the original image has been done and the results are shown in Table 4.1 and Table 4.2 for the two undersampling configurations. These tables show that even though the raster pattern outperform at the two lowest levels, the PSNR and SSIM is still at an acceptable high level.

Since the goal of the project is to improve speed, it is of interest how long the

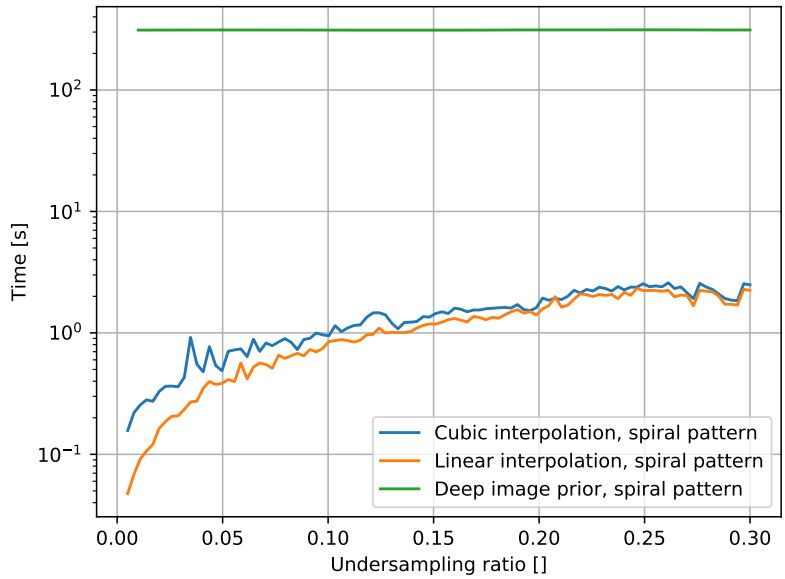


Figure 4.3: Reconstruction time versus undersampling ratio for baseline and DIP.

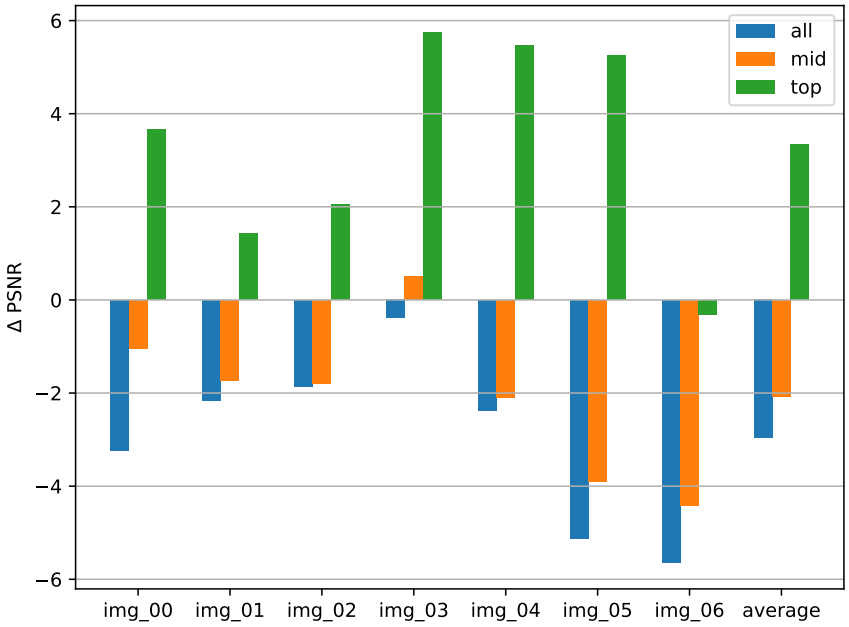


Figure 4.4: Relative PSNR compared to a sample pattern of the same length. The undersampling ratios are $[0.02, 0.05, 0.10]$.

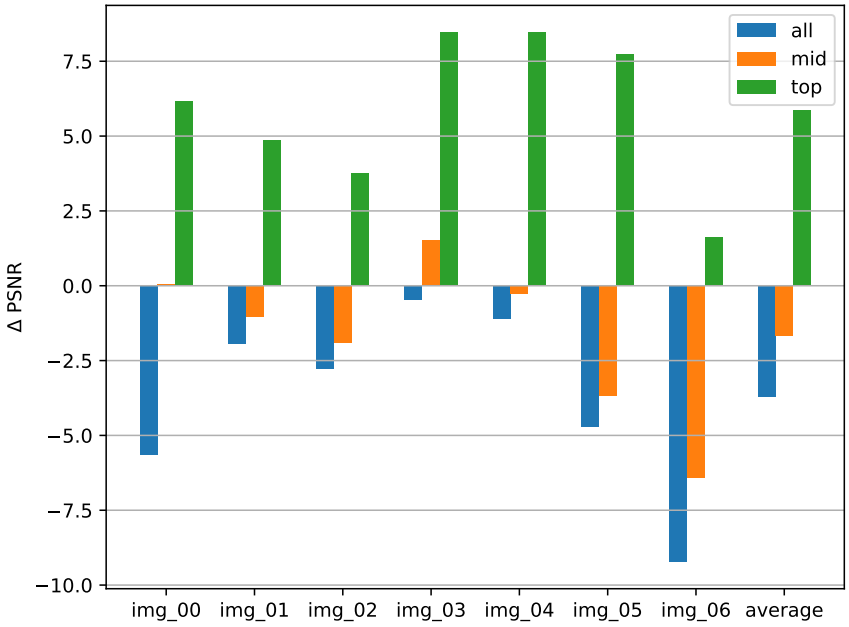


Figure 4.5: Relative PSNR compared to a sample pattern of the same length. The undersampling ratios are $[0.02, 0.10, 0.20]$.

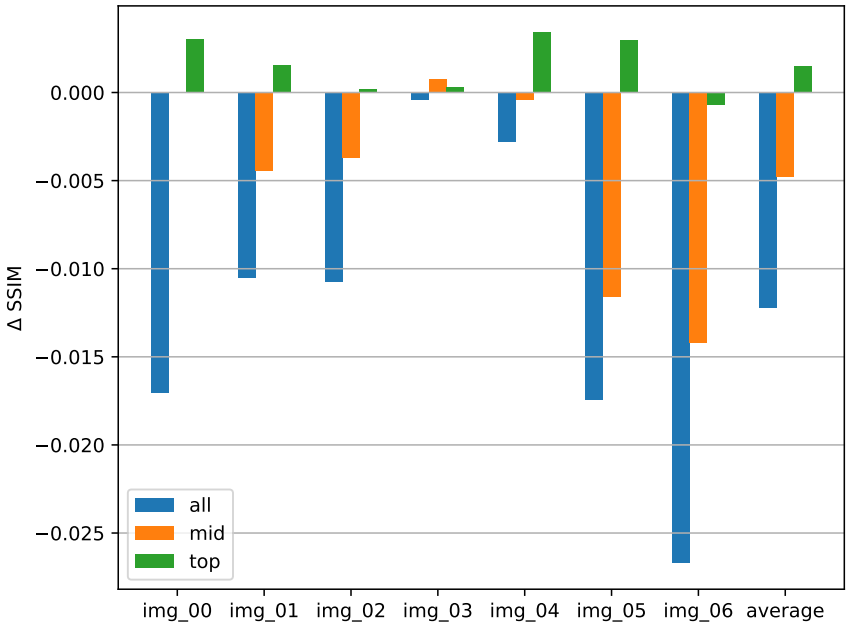


Figure 4.6: Relative SSIM compared to a sample pattern of the same length. The undersampling ratios are $[0.02, 0.05, 0.10]$.

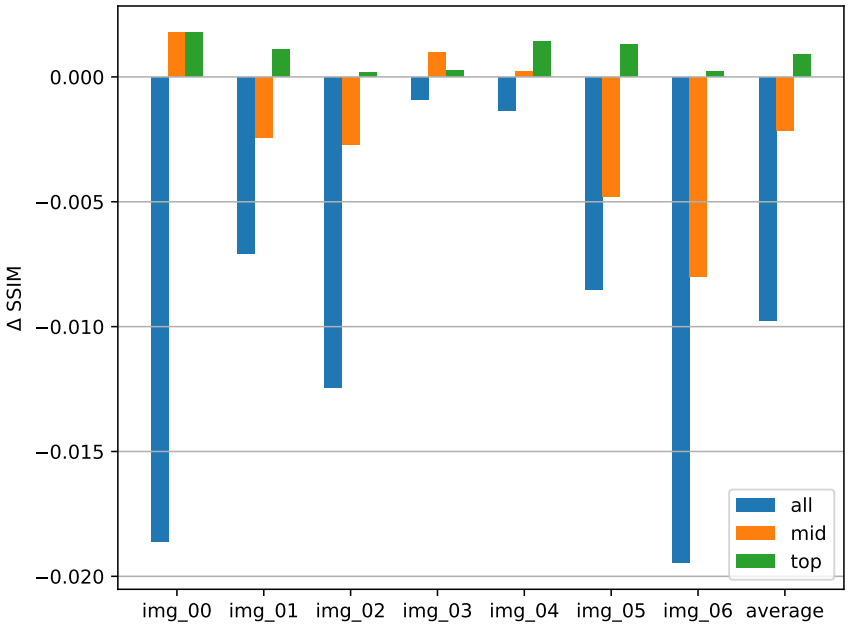


Figure 4.7: Relative SSIM compared to a sample pattern of the same length. The undersampling ratios are $[0.02, 0.10, 0.20]$.

Table 4.1: PSNR and SSIM for the three layers. Undersampling ratio of [0.02, 0.05, 0.10].

	PSNR [dB]			SSIM		
	All	Mid	Top	All	Mid	Top
image_00	43.2	46.9	58.3	0.941	0.973	0.997
image_01	39.3	40.7	49.7	0.962	0.976	0.997
image_02	41.0	43.7	56.0	0.957	0.986	0.999
image_03	46.4	49.9	65.0	0.985	0.995	0.999
image_04	42.0	42.6	54.6	0.98	0.984	0.998
image_05	39.5	41.2	54.8	0.966	0.974	0.998
image_06	40.0	41.8	50.0	0.959	0.974	0.996
Average	**41.6**	**43.8**	**55.5**	**0.964**	**0.980**	**0.998**

Table 4.2: PSNR and SSIM for the three layers. Undersampling ratio of [0.02, 0.10, 0.20].

	PSNR [dB]			SSIM		
	All	Mid	Top	All	Mid	Top
image_00	44.9	52.4	65.2	0.962	0.99	0.999
image_01	43.6	45.5	57.5	0.981	0.989	0.999
image_02	43.2	46.9	62.1	0.969	0.992	0.999
image_03	49.2	54.1	70.8	0.990	0.997	0.999
image_04	48.3	49.5	62.6	0.992	0.994	0.999
image_05	45.2	46.7	62.6	0.985	0.99	0.999
image_06	42.7	46.0	58.3	0.976	0.989	0.999
Average	**45.3**	**48.72**	**62.72**	**0.979**	**0.992**	**0.999**

reconstruction takes. In Table 4.3 and Table 4.4 the reconstruction time and the speedup is shown for the two undersample ratio configurations.

The speedup is the total length of the adaptive sample pattern. An estimate of the runtime for these patterns can be calculated using $t_{\text{adaptive}} = t_{\text{raster}} \cdot \text{speedup}$ e.g. a raster scan which takes 3 minutes with a speedup factor of 0.1 the resulting time would become 18 seconds which is a significant speedup. This also makes the reconstruction time of a few seconds negligible.

Table 4.3: Reconstruction time and speedup. Undersampling ratio of [0.02, 0.05, 0.10].

	Reconstruction time [s]	Speedup
image_00	5.45	0.090
image_01	6.56	0.094
image_02	4.89	0.073
image_03	4.12	0.068
image_04	3.82	0.076
image_05	5.04	0.080
image_06	5.07	0.106
Average	**4.99**	**0.084**

Table 4.4: Reconstruction time and speedup. Undersampling ratio of [0.02, 0.10, 0.20].

	Reconstruction time [s]	Speedup
image_00	8.59	0.149
image_01	12.6	0.151
image_02	8.44	0.116
image_03	6.23	0.104
image_04	5.8	0.119
image_05	7.41	0.128
image_06	7.84	0.181
Average	**8.13**	**0.135**

5 DISCUSSION AND CONCLUSION

In this chapter the results of the project will be discussed and concluded. In the process of this, the pros and cons of the used methods is presented. Additionally some possible improvements are proposed.

The results from chapter 4 shows the performance of DIP, bicubic and bilinear interpolation, the adaptive sampling pattern and the speedup in time.

From Figure 4.1 and Figure 4.2 it is seen that DIP performs worse than bicubic and bilinear interpolation. This is speculated to be because the stopping criterion is fixed at 2000 iterations. DIP is theorized to perform better by increasing the number of iterations. However, this is not considered since the time consumption of DIP is significantly longer than that for interpolation as shown in Figure 4.3 and thus makes it unfit for this project and speedup of AFM.

The performance of the adaptive sampling pattern is evaluated both by comparing it to a raster pattern of equal length and to the original image. The results show that the adaptive pattern is outperformed in the whole image and has approximately the same performance for the important area. However, the top layer has significantly better PSNR and also performs better for SSIM. Even though the adaptive sampling pattern is outperformed at the lower layers, the reconstruction quality of all the layers are still deemed of acceptable accuracy with a PSNR of more than 39.3 dB and a SSIM of above 0.941 compared to the original image. This is shown with the two undersampling ratio configurations tested in Table 4.1 and Table 4.2 but is also showed in the attached material. The results are satisfying since the goal has been to focus on the regions of interest in the image. Through the project the top layer has been considered the most important part of the image an thus it is important that this region is of high quality. The reconstructed image has an average speedup of 10 times for the reconstruction in the mid layer with a performance of 44 dB PSNR.

The concept in this project is mostly suitable for large images which has a scanning time from approximately 3 minutes to 17 minutes, as described in chapter 1. This makes the reconstruction time present in Table 4.3 and Table 4.4 of few seconds negligible. However, this reconstruction time might be improved by implementing the solution in a more low level programming language such as C.

Other ways of detecting the regions of interest could be interesting for further work. One method to detect the regions could be to apply a moving variance filter to the sparse representation of the image. This would give an estimate of the regions with high information density and thus give an idea of where the interesting parts of the image occur. These regions would then be sampled in the same way as done in this project.

As for detecting the regions of interest it might also be interesting to use other reconstruction algorithms with the sampling pattern achieved in this project. This could be an algorithm like TV or Soft Iterative Thresholding, as these methods have quick reconstruction times and also perform well like interpolation. There might be a reconstruction algorithm that, when paired with the adaptive sampling pattern, could achieve even better results.

Based on this, it is concluded that the proposed method in this project is a feasible solution for speeding up the scanning time in AFM while keeping the relevant parts of

the image at high resolution and quality. However, it is also concluded that DIP is not suitable due to it's slow reconstruction time.

BIBLIOGRAPHY

[1] Y. Luo and S. B. Andersson, "A continuous sampling pattern design algorithm for atomic force microscopy images," *Ultramicroscopy*, vol. 196, pp. 167 – 179, 2019.

[2] T. Arildsen, C. S. Oxvig, P. S. Pedersen, J. Østergaard, and T. Larsen, "Reconstruction algorithms in undersampled afm imaging," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 1, pp. 31–46, Feb 2016.

[3] K. Zhang, T. Hatano, T. Tien, G. Herrmann, C. Edwards, S. C. Burgess, and M. Miles, "An adaptive non-raster scanning method in atomic force microscopy for simple sample shapes," *Measurement Science and Technology*, vol. 26, no. 3, p. 035401, feb 2015.

[4] Y. Wen, J. Song, X. Fan, D. Hussain, H. Zhang, and H. Xie, "Fast specimen boundary tracking and local imaging with scanning probe microscopy," *Scanning*, vol. 2018, pp. 3 979 576–3 979 576, Mar 2018, 29692874[pmid].

[5] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Deep image prior," *CoRR*, vol. abs/1711.10925, 2017.

[6] B. B. (eds.), *Springer Handbook of Nanotechnology*, 4th ed., ser. Springer Handbooks. Springer-Verlag Berlin Heidelberg, 2017.

[7] C. Rankl, "Atomic force microscopy images of cell specimens," May 2015. [Online]. Available: https://doi.org/10.5281/zenodo.17573

[8] C. S. Oxvig, P. S. Pedersen, J. Østergaard, T. Arildsen, T. L. Jensen, and T. Larsen, "Magni," Website, 2018. [Online]. Available: https://github.com/SIP-AAU/Magni

[9] Scikit-image development team, "Scikit-image, module: measure," Website. [Online]. Available: https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label

[10] R. Fisher, S. Perkins, A. Walker, and E. Wolfart., "Pixel connectivity," Website, 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/connect.htm

[11] C. Bishop, *Pattern recognition and machine learning*, M. Jordan, J. Kleinberg, and B. Schölkopf, Eds. Springer-Verlag New York, 2006.

[12] scikit-learn developers, "Clustering," Website, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#k-means

[13] J. Dahl, P. C. Hansen, S. H. Jensen, and T. L. Jensen, "Algorithms and software for total variation image reconstruction via first-order methods," *Numerical Algorithms*, vol. 53, no. 1, p. 67, Jul 2009.

[14] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259 – 268, 1992.

[15] A. Chambolle, "An algorithm for total variation minimization and applications," 2004.

[16] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Deep image prior, supplementary material," 2017.

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354 EP –, Oct 2017, article.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[20] D. Salomon, *Data Compression: The Complete Reference*. Springer, 2006.

[21] DmitryUlyanov, "Dmitryulyanov/deep-image-prior," Website, 2017. [Online]. Available: https://github.com/DmitryUlyanov/deep-image-prior