Multiscale spatial modeling with applications in image analysis

by

Carel van Niekerk

13013492

Submitted in partial fulfillment of the requirements for the degree

Magister Scientiae

In the Department of Statistics

In the Faculty of Natural and Agricultural Sciences

University of Pretoria

12 November 2018

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Declaration

I, *Carel van Niekerk,* declare that this mini-dissertation (100 credits), which I hereby submit for the degree Magister Scientiae in Mathematical Statistics at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

Signature:

Date:

Acknowledgements

I would firstly like to thank God for giving me the health, power and talents to be able to complete this research

I would then like to express my gratitude towards my supervisor Dr. Inger Fabris-Rotelli for her invaluable support and motivation during this research. Without her motivation, patience and positivity I would not have been able to complete this work. Her comments and input in this research was invaluable to me. I do not think I could ever get a better supervisor.

I also thank my co-supervisor Dr. van Niekerk for her valuable inputs and encouragement to better this research.

I also thank the National Research Fund SASA grant, the statistics HUB University of Pretoria, the Department of Statistics University of Pretoria and the Centre for artificial intelligence research at the CSIR for their financial support during this work. All the views is this report is my views and not those of my financial sponsors.

I also thank my friends at the statistics HUB at the University of Pretoria for their support. Without their jokes and all our fun unproductive times I would not have remained sane.

Last but not least I would like to thank my parents Carel and Celeste and my brother Allan and the rest of my family for all their support during my studies. I would also like to thank my parents for their financial support during my studies. I also thank my girlfriend Claudia for her amazing support and encouragement when I thought I cannot do this any more I also thank her for her language edits in this work.

Contents

List of Figures

List of Tables

Abstract

Computer vision is a very important research area and is continuously growing. One of the prevalent research areas in computer vision is image matching. In image matching there are two main components, namely feature detection and feature matching. The aim of this this study is to determine whether Direct Sampling can be used for feature matching, and also if the combination of Direct Sampling and the Discrete Pulse Transform feature detector can be a successful image matching tool. In feature detection there are many strong methods including convolutional neural networks and scale-space models such as SIFT and SURF, which are very well-known feature detection algorithms. In this work we utilize another scale-space decomposition tool called the Discrete Pulse Transform (DPT). We particularly use the DPT decomposition to enable significant feature detection. We then concentrate on using the Direct Sampling algorithm, a stochastic spatial simulation algorithm, for modelling and matching of features. We do not consider convolutional neural networks or SIFT or SURF for texture matching in this work, this is because we particularly focus on the use of spatial statistics in image matching. We finally propose a novel multiscale spatial statistics feature detection and matching algorithm which combines the DPT feature detection with Direct Sampling for feature matching, specifically for texture classes of images. The performance of the proposed method is tested by comparing the distances obtained from the proposed algorithm between different texture images. We see that this proposed novel multiscale spatial modelling approach to feature matching with the focus on textures performs well at discriminating between difficult to discriminate between textures.

Chapter 1

Introduction

Computer vision is a very important area of artificial intelligence research. This field of research has grown immensely, and continues to. Prevalent problems in computer vision include tracking an object in a video sequence. Another problem artificial intelligence faces is finding visually similar areas in images such as a human for a self-driving car. This type of problem can be solved using an approach called image matching. Image matching is described by Dai and Lu [9] as 'the process of bringing two images geometrically into agreement so that corresponding pixels in the two images correspond to the same physical region of the scene being imaged'. Image matching is a significant tool in various disciplines. In satellite imagery it often occurs that one needs to overlay different images of the same area to investigate whether some change has occurred. To overlay these images we need to match the corresponding areas in the different images [17]. When we want to recognise a object in an image we can match the features in the image to features in images with labels, which are known at outset [3]. For instance, a robot that needs to localise itself in an environment can match features in what it sees to features of known locations. This help the robot find its location [57]. Another example is object tracking, where we match the features of the object being followed to features in the video sequence frames. This will assist in locating and tracking the object in the video [30]. These are just some examples of the wide range of application for image matching.

For image matching there are two main approaches, global and local matching. When using global image matching the image as a whole is summarised, generally using some form of dimension reduction or encoding. A similarity score or distance between different images is then calculated using the encoding. Local image matching concentrates on determining feature regions in an image and attempts to match these to similar regions in other images. Local matching can also be used as a method for global matching by summarizing a collection of local matches.

In local feature matching there are generally two steps involved. Firstly the image is segmented to extract significant features. Feature detection is the process of segmenting an image into disjoint and significant segments called features [48]. Secondly, the features are matched to features in another image

using a feature localisation process. Feature localisation is the process of finding a region or segment which is visually similar to a feature.

1.1 Stochastic spatial simulation and feature matching

The field of spatial statistics can be well described in general by Tobler's First Law of Geography which states that 'Everything is related to everything else, but near things are more related than distant things' [56]. We propose here to extend these ideas of spatial statistics to image matching, using the spatial relationship between pixels more accurately.

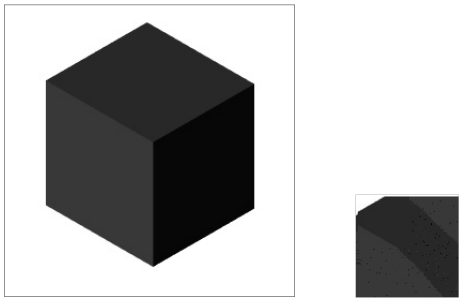
A set of spatial statistics algorithms which can be applicable in this field are known as stochastic spatial simulation algorithms. A subset of stochastic spatial simulation algorithms is Multiple-Point Statistics (MPS) which simulates spatial points directly from a training dataset [55]. Some of the well known MPS algorithms are single normal equation simulation (SNESIM) [53] or extended normal equations simulation (ENESIM) [18]. During 2009 Google acquired a patent on SNESIM for spatial information or image simulation [54]. Direct Sampling (DS) [32] is also a MPS algorithm. The prominent difference between Direct Sampling and SNESIM or ENESIM is that DS simulates directly from a training image, whereas SNESIM and ENESIM model probability distributions of a training dataset [55].

Direct Sampling simulates directly from the training image by using a scanning window/filter. This window consists of the neighbourhood of the point to be simulated. The point in the training image with the most similar neighbourhood is then found using this window. This point is then placed at the position of the point being simulated. In Figure 1.1 is an example of an image simulated using Direct Sampling. The simulation in 1.1 b) is a simulation of the spatial relationships between the pixels in 1.1 a). This simulation is obtained using the procedure described in Chapter 2.

Direct Sampling was developed to model and simulate textures in images. In this research it is further investigated whether Direct Sampling can be extended to be used as a feature matching method for textures. We aim to do this by not using DS as a simulation method but rather as a method which links similar pixels in two images.

1.2 Multiscale methods for feature detection

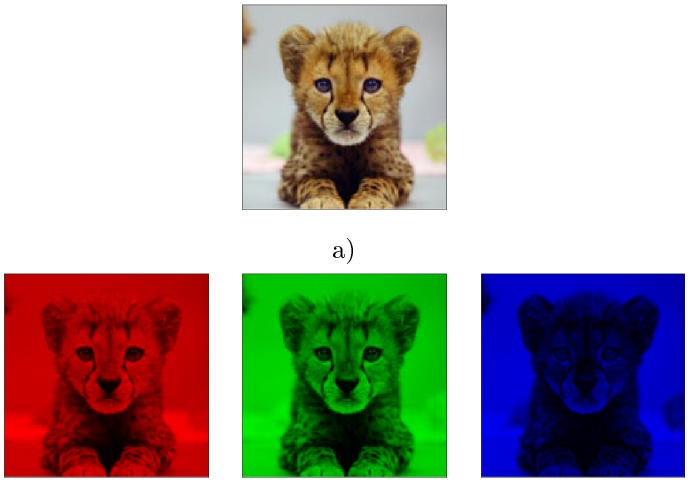
Some of the state of the art image matching algorithms that are discussed in literature are the Scale-Invariant Feature Transform (SIFT) [29], the Speeded-Up Robust Features (SURF) [6] and the Oriented-FAST and Rotated BRIEF (ORB) [46]. In [17] a discussion of these multiscale algorithms is provided. A multiscale decomposition of a signal results in multiple scale layers. Each of these layers represents structures in the signal at the respective scale [10]. Scale-space theory is a structure with which the multiscale nature in images can be portrayed [25]. It is important to not confuse multiscale with multidimensional or multiresolutional. Multidimensional images are images where, for example the colour information in



a) Training Image b) Simulation

Figure 1.1: Illustration of Direct Sampling image simulation. a) Image of a box and b) Simulation of this image using Direct Sampling.

an image is stored in multiple bands (dimensions). Some of the most common multidimensional image formats are RGB (Red, Green, Blue), CMY (Cyan, Magenta, Yellow), HSV (Hue, Saturation, Value), RGB-D (RGB with a depth dimension), and multispectral. Multispectral images capture image data within specific wavelength ranges such as visible, ultra-violet or infrared light. In Figure 1.2 the different dimensions of an RGB image are illustrated. From this illustration it can be seen that these different dimensions are not representations at different scales but simply the colour dimensions which collectively form the colour image.



a)

b) c) d)

Figure 1.2: An RGB image multidimensional decomposition. a) RGB Image, the red dimension in b), green in c), and blue in d).

Multiresolutional images consider the same image at different resolutions. These images are most often considered as pyramids, where each image is reduced by an octave to form the next level. This basically

means that with each octave reduction groups of four pixels are combined into one, hence reducing the resolution of the image [2]. An example of such an image pyramid is provided in Figure 1.3. It is argued

Figure 1.3: An illustration of a multiresolutional pyramid, constructed from the image on the left. Each of the images on the right is a one octave reduction from the previous image in the pyramid.

that multiresolutional images help achieve scale invariance, due to the fact that reducing the resolution will also reduce the scale at which objects are portrayed [29]. Scale invariance is the ability to recognise a feature irrespective of its scale.

In [26] Lindeberg demonstrates that Gaussian multiscale interest point detectors are advantageous. In this research we will be using a scale-space method, other than that used by Lindeberg, to extract multiscale regions of interest in an image. These regions can then be used for matching.

When considering image matching and feature detection, neural networks should also be considered. Convolutional neural networks (CNN) are effective tools for computer vision tasks such as classification of images [50, 20]. Zeng and Zhu proposed a novel multiscale fully convolutional network for foreground object detection [61]. The convolutional architecture used by Zeng and Zhu is a multiscale tool, because convolutional layers of selected scales are passed over the image consecutively. These layers extract a selected scale-space from the image. We call this a selected scale-space, because the CNN only extracts scale layers of the scales selected. Other scale-space algorithms, like the DPT, extract scale layers of all possible scales. This produces a perfect full decomposition as well as a perfect reconstruction.

Li and Yu use multiscale deep features extracted using a CNN to determine visual saliency [23]. A visually salient feature is one that is visually noticeable to the human eye. It can also be described as the visual contrast between a feature and its neighbours [23]. Visual saliency is often used for computer vision tasks such as object recognition [47, 60, 62]. Some of the datasets often used to test visual saliency algorithms are MSRA-B [28], SED [4] and SOD [33].

The well known SIFT and SURF algorithms uses a multiscale structure called the Gaussian scale-space. Gaussian scale-space structures are extracted by smoothing the image using Gaussian filters at selected scales. Lindeberg provides the basic ideas of Gaussian scale-spaces in his book [25]. To explain Gaussian scale-spaces we will illustrate them in a short example. Starting with the multivariate Gaussian density function which is defined as follows. For a filter centred at $\mathbf{c} = (c_1, c_2)$ with scale s, the value of

the filter at point $\mathbf{x} = (x_1, x_2)$ is

$$g(\mathbf{x}; \mathbf{c}, s) = \frac{1}{2\pi s} e^{-\frac{(x_1-c_1)^2+(x_2-c_2)^2}{2s}}.$$

An example of this is filter is shown in Figure 1.4. An image is smoothed using this filter by convoluting the filter across the image. Convoluting the filter across the image means that some function of the filter and the image is stored for point in the image. To perform smoothing, the function used is the weighted sum of all the neighbours of a point $\mathbf{x}$. The weights used are the values of the filter centred at $\mathbf{x}$. The convolution obtained, $f(\mathbf{x}, s) = f(\mathbf{x}) \circ g(\mathbf{x}; \mathbf{c}, s)$, is the new smoothed image. The application of the above process can be seen in Figure 1.5. The sum of the values at all the points in Figure 1.5 c) becomes the new smoothed value of the pixel $\mathbf{x} = (100, 100)$. Obtaining the smoothed value of each pixel using the Gaussian filter, named convoluting the Gaussian filter across the image, results in the smoothed image in Figure 1.5 d).

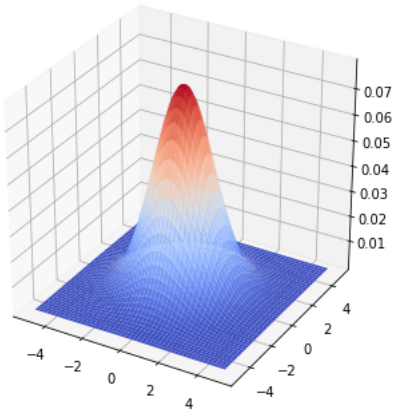
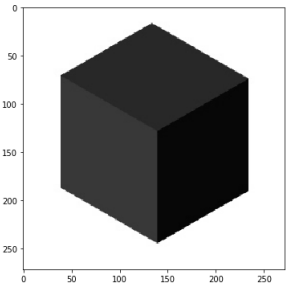


Figure 1.4: Gaussian Filter centred at (0,0) with scale 2.

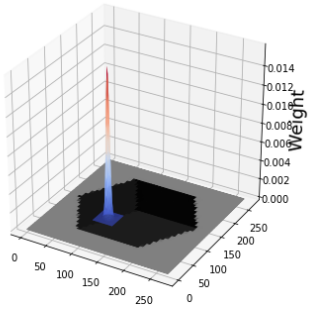
The Gaussian scale-space is also used to find interest points in an image. The method used for this is called the Hessian affine region detector, described in [36]. In this approach the Hessian matrix, the matrix of second derivatives of the smoothed image, is used to find interest points. The Hessian matrix of a smoothed image $f(\mathbf{x}, s)$ at pixel $\mathbf{x} = (x_1, x_2)$ is defined in Equation 1.1.

$$H(\mathbf{x}, s) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(\mathbf{x}, s) & \frac{\partial^2}{\partial x_1 \partial x_2} f(\mathbf{x}, s) \\ \frac{\partial^2}{\partial x_2 \partial x_1} f(\mathbf{x}, s) & \frac{\partial^2}{\partial x_2^2} f(\mathbf{x}, s) \end{bmatrix} \tag{1.1}$$

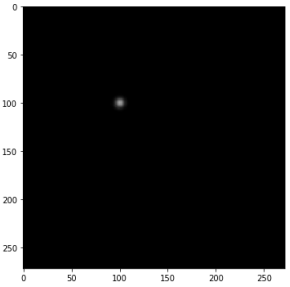
The Hessian matrix of each point in the signal is determined by convoluting each of the 4 second derivatives shown in Figure 1.6 over the image. When convoluting these filters over the image we obtain the Hessian matrix $H(\mathbf{x}, s)$ of the signal at each point. We then determine the determinant of the Hessian matrice, and use the well known mathematical result which is described in [51] to find the local minimums or maximums, using the determinants. This points out regions in the image where the points are significantly



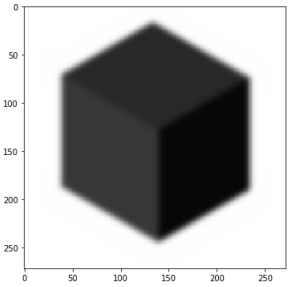
a) $f(\mathbf{x})$



b) $g(x, (100, 100), 10)$



c) $f(\mathbf{x}) \times g(x; (100, 100), 10)$



d) $f(\mathbf{x}, s) = f(\mathbf{x}) \circ g(\mathbf{x}; (100, 100), 10)$

Figure 1.5: Illustration of Gaussian Smoothing. In a) we have an image of a box. In b) we have the Gaussian filter of scale 10 centred at pixel $\mathbf{x} = (100, 100)$. In c) we see the result of the filter in b) multiplied by the image pixelwise. The resulting smoothed image is in d).

different from their neighbours. This yields interest points of scale s in the image. This method is applied iteratively over a selected range of scales to obtain interest points at a selection of scales.

SURF, developed by Bay et.al. [6], is a well-known feature detection and matching algorithm. The Hessian affine region detector is used by SURF, hence SURF can be considered a multiscale tool. SURF is efficient due to the following reasons. SURF is scale invariant because it uses filters of multiple scales to detect feature points. This means it will detect a feature point regardless of its scale. SURF is also location invariant because the filters are sliding windows, performing the same operations at each pixel. Therefore it will detect feature points regardless of their location in the image. The aim with the algorithm is to be as fast and efficient as possible whilst maintaining an acceptable accuracy. The accuracy obtained is not the ultimate optimal accuracy due to the approximations used, but it is close enough to be acceptable.

The first part of SURF is to identify interest points. These interest points are identified using an approximation of the Hessian affine region detector; box filters are used to approximate the Hessian affine method. Box filters approximate the second derivatives illustrated in Figure 1.6, approximating local extrema using constant boxes. In Figure 1.7 we show the approximations of the Gaussian second derivative filters.

Bay et al. [6] also state that commonly interest points of different scales are extracted from an

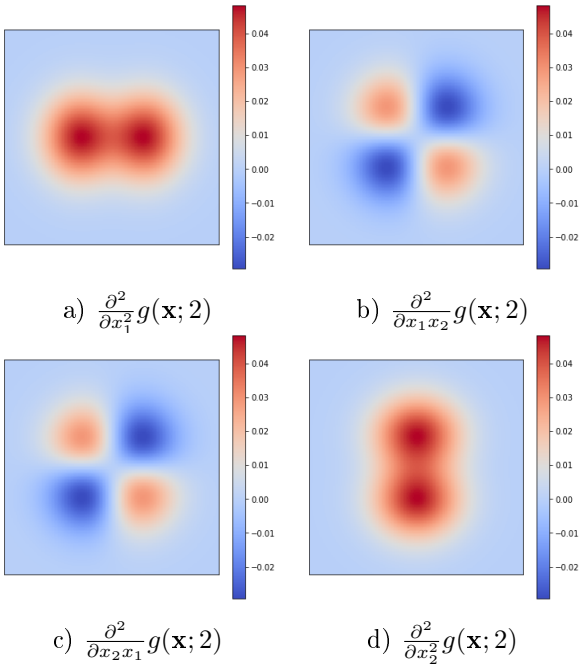


Figure 1.6: Second derivatives of the Gaussian filter $g(\mathbf{x}; 2)$.

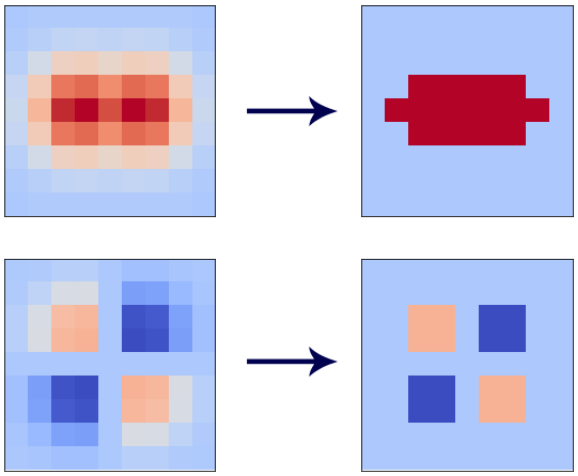


Figure 1.7: Illustration of box filter approximations for second derivatives of Gaussian filter.

image pyramid to save time. However the box filters used by SURF work equally fast on the full image, irrespective of the scale. Hence a pyramid is not necessary [6]. The extracted interest points are then described using modified Haar wavelet distributions. According to [6] SURF performs efficiently and accurately because it is simplified enough to reduce computation costs but still perform accurately enough. As mentioned, some accuracy is lost to make the algorithm computationally efficient. In Figure 1.8 we can see an example of interest points detected by SURF.

Two other well known scale-space models are the DoG (difference of Gaussians) [24, 29] and LoG

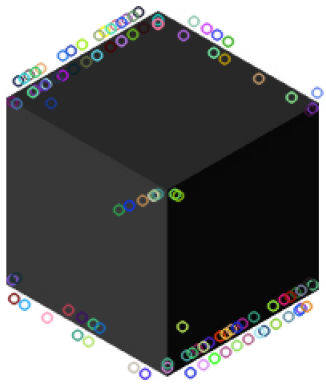


Figure 1.8: Illustration of the interest points extracted by SURF.

(Laplacian of Gaussians) [24, 35] filters. The LoG and DoG models are commonly used for edge detection in images. Since DoG by SIFT, we will focus on DoG. According to [16] many edge enhancement algorithms enhance high frequency details, which could result in the undesired enhancement of noise. DoG extracts edges/details by taking the difference between two smoothed images, each smoothed at a different scale. This ensures that DoG does not enhance high frequency details, such as noise. The scales s and s_1, used by DoG, are defined as $s_1 = sk^2$ where k is the size ratio. A selection of different size ratios is commonly used to extract edges or details at different scales. The DoG filter is defined as follows [26]

$$DoG(\mathbf{x}; \mathbf{c}, s, k) = \frac{1}{s(k-1)} \left(g\left(\mathbf{x}; \mathbf{c}, sk^2\right) - g(\mathbf{x}; \mathbf{c}, s) \right).$$

In Figure 1.9 an illustration of a DoG filter is given. In Figure 1.10 it is illustrated how the DoG filter enhances edges. These edges are enhanced or extracted by taking the difference between two smoothed images, with the two images smoothed by the Gaussian filter at different scales. This method works because details are lost whilst smoothing an image, meaning the difference between two images smoothed at different scales will be those lost details. This method is a multiscale tool because it uses multiple scales to detect details.

SIFT (Scale Invariant Feature Transform) is a well known and efficient feature detection and matching algorithm. It was developed by Lowe [29] in 2004. SIFT uses image pyramids and DoG filters [24] for feature extraction contrary to the box filters used by SURF. The SIFT feature extraction tool is a multiscale tool because it uses difference of Gaussian scale-spaces. SIFT is a fast feature extraction algorithm which is scale, location and rotation invariant. This implies that an object will be detected accurately irrespective of its scale, location and rotation. This is achieved by using the scale and location invariant DoG filters, in combination with image pyramids and multiple orientations of features. One of the first steps followed by SIFT, to reduce computation cost, is to use cascade filtering. At this stage tests are performed to determine whether a computationally expensive operation should be carried out.

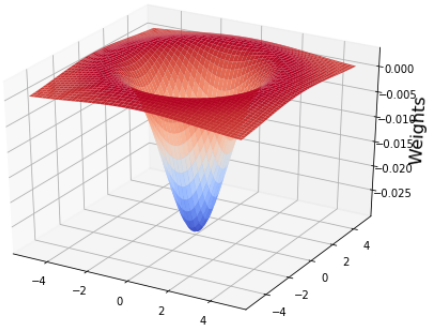


Figure 1.9: Illustration of a difference of Gaussians filter with scale $s = 2$ and size ratio $k = 2$.

a) b)

Figure 1.10: Edge enhancement by difference of Gaussians (DoG). In a) is the original image and in b) the details extracted by the DoG filter.

In Figure 1.11 we can see an example of interest points detected by SIFT. The majority of these interest points detected are edges or corners which are salient details which can be used to match features. Other uses of multiscale models in feature detection are further discussed by Lindeberg [27].

We will now look at the prominent differences between the Discrete Pulse Transform (DPT) [5] used in this research and the SIFT and SURF algorithm described in this section. The DPT decomposition is a collection of pulses such that an image can be represented by the sum of this collection of pulses $\{p_{ni}(f)\}$ as follows:

$$f(\mathbf{x}) = \sum_{n=1}^{N} D_n(f)(\mathbf{x}) = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} p_{ni}(f)(\mathbf{x}).$$

Each $D_n(f)$ is a scale layer of scale n made up of all the pulses of that scale, $\{p_{n1}(f), p_{n2}(f), ..., p_{n\gamma(n)}(f)\}$, where $\gamma(n)$ is the number of pulses of scale n. The DPT is adaptive to shape, due to the shape adaptable neighbourhoods used while filtering. SIFT and SURF uses fixed shape filters, this means they cannot adapt

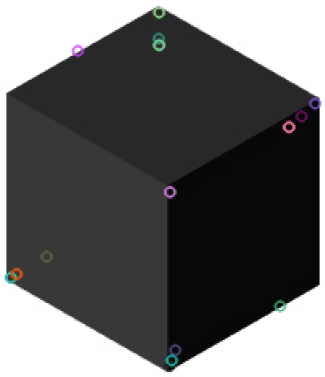


Figure 1.11: Illustration of the interest points extracted by SIFT.

to shape. SIFT and SURF, and any other Gaussian scale-space methods, use a continuous filter discretized for use on discrete digital images. The DPT provides a naturally discrete tool for image analysis. The DPT smoothens a signal by flattening local minimums and maximums. These local minimums and maximums are identified using the neighbourhoods in the signal. These neighbourhoods are discretely defined and can be adapted to any shape required. In Chapter 3 we discuss the DPT in more detail. We also illustrate there the ability of the DPT to extract features. The use of the DPT as a feature detection algorithm has been discussed in literature such as [58, 12, 52, 5]. Although a lot of work has been done there is still more work to be done. Some of the work that still needs to be done in this work involves improving the computational efficiency. We also discuss the creation of feature maps and pulse maps, maps which contain only significant parts of a reconstruction. In Figure 1.12 we can see an example of interest points detected by the DPT. These interest points are extracted using a significance score which is calculated using the DPT decomposition of the image, which is discussed in more detail in Chapter 3. We can see that the DPT feature detection has detected more salient edge information than the SIFT algorithm in Figure 1.11. The DPT feature detection method has also performed similarly to the SURF algorithm in Figure 1.8, not in terms of the exact information detected but in terms of the number of salient features detected. It is important to note that we will not be utilising the SIFT and SURF methods in this work. We particularly focus on the use of spatial statistics for feature matching.

1.3 Layout

In this mini-dissertation we aim to show the following:

- Test whether Direct Sampling can perform the role of a feature localisation and image matching algorithm for textures. This will be a new use of the stochastic simulation algorithm.
- Propose a novel multiscale image matching algorithm which combines the DPT feature extraction

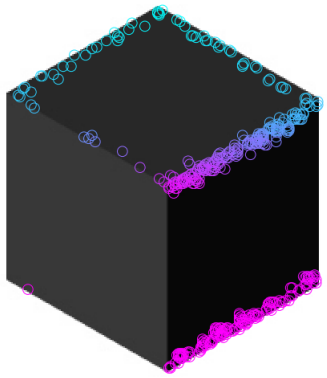


Figure 1.12: Illustration of the interest points extracted by the DPT.

algorithm with the DS feature matching algorithm.

- Test this new multiscale feature matching algorithm's performance.

In Chapter 2 we discuss stochastic spatial simulation, and in particular the Direct Sampling (DS) algorithm. We then investigate the faster bunch-pasting DS implementation. We also discuss a new implementation of the Direct Sampling algorithm as a feature localisation and image matching method. In Chapter 3 we discuss the scale-space theory of the Discrete Pulse Transform (DPT). We discuss the use of the Roadmaker's pavage (RP) implementation, a faster graphical DPT decomposition algorithm. We then discuss feature detection and a new graphical DPT feature extraction method. This method is based on the RP graphs and feature detection work done by van der Walt [58]. We also discuss a texture extraction method proposed and used by Fabris-Rotelli and Stein [10, 11]. In Chapter 4 we propose a novel feature detection and matching algorithm which uses the DPT feature extraction and DS matching for textures. We test this method using the USC-SIPI texture database [1]. Finally in Chapter 5 we summarise our results and make final concluding remarks, as well as highlight areas for future research.

Chapter 2

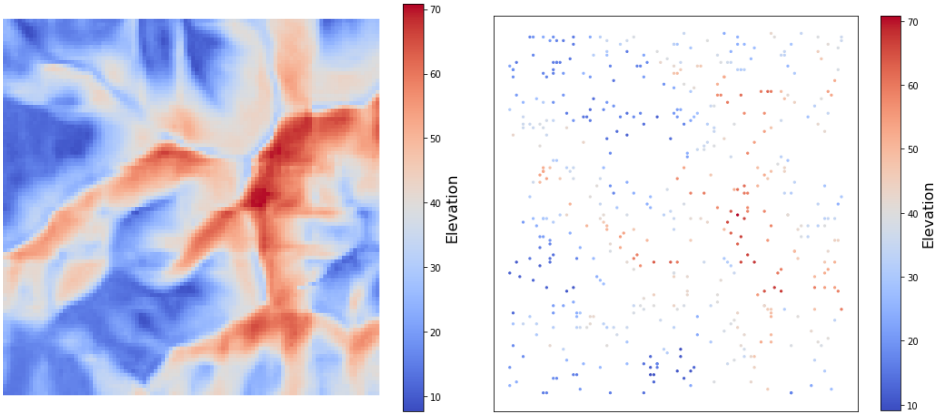
Feature matching using Direct Sampling

In Chapter 1 we briefly discussed the use of Direct Sampling for feature matching. In this chapter we discuss in more detail stochastic spatial simulation and Direct Sampling. We also illustrate the use of Direct Sampling, in particular the bunch-pasting version, as a feature matching method.

2.1 Introduction

In Chapter 1 we discussed an area of spatial statistics with which we can simulate images, called stochastic spatial simulation. An example of a typical problem in stochastic spatial simulation is finding the elevations of a mountain for planning of new access routes [31]. Due to logistical reasons elevation measures can only be taken at 500 points on the mountain. An example elevation map of this mountain is shown in Figure 2.1 a). In Figure 2.1 b) we see the random sample of 500 points taken.

A solution to this problem is to spatially simulate (estimate) the remaining elevations of this mountain. Traditionally methods such as sequential indicator simulation using variograms or object-based simulation methods were used [55]. In 1951 one of the more popular methods named Kriging [19] was developed by South African mathematician Danie Krige. Some of the other well known random function theory methods are single normal equation simulation (SNESIM) [53] and extended normal equations simulation (ENESIM) [18]. These methods simulate from a training dataset consisting of images, and rely on Gaussian theory. This set of methods is called multiple point geostatistics (MPS) algorithms. MPS is a framework in which a range of complex structures in spatial fields can be modelled [34]. SNESIM has already played a role in the image simulation and matching area when it was patented by Google in 2009 [54]. One of the drawbacks of these models is that they model a probability distribution. This task is very computationally intensive [31]. Direct Sampling was developed to be more computational efficient by simulating directly from a training image. We will now further discuss Direct Sampling (DS) in detail. The Direct Sampling algorithm is based on the following language communication model proposed by Shannon in 1948 [49].



a) b)

Figure 2.1: Illustrated is a random sample from a spatial elevation map. In a) is the true spatial elevation map (screen shot taken from `https://www.google.com/maps`) and in b) is a random sample of elevation points.

2.1.1 Early development of direct sampling techniques

The approach of [49] is to sample directly from text. The algorithm follows the following steps: First select a random character from the text, say 'k'. Then select a random paragraph in the text, and find the first occurrence of the character 'k'. Then store the character that occurred immediately after 'k', say 'e', resulting in 'ke'. The above process is then repeated to obtain, say 'ker'. This process is repeated until we have sampled N characters. During this process we only consider the previous L characters when searching. This is done since we might not find a exact match to the simulated text.

As an example lets consider the following extract from 'The Ransom of Red Chief' by O. Henry (this text is available at `http://learningenglish.voanews.com/a/a-23-2009-06-12-voa1-83142177/117086.html`):

'*Bill and me had a joint capital of about six hundred dollars, and we needed just two thousand dollars more to pull off a fraudulent town-lot scheme in Western Illinois with. We talked it over on the front steps of the hotel. Philoprogenitiveness, says we, is strong in semi-rural communities therefore, and for other reasons, a kidnapping project ought to do better there than in the radius of newspapers that send reporters out in plain clothes to stir up talk about such things. We knew that Summit couldn't get after us with anything stronger than constables and, maybe, some lackadaisical bloodhounds and a diatribe or two in the Weekly Farmers' Budget. So, it looked good.*'

Let's consider a random character 'j'. We find the first occurrence of 'j' in this text, which is at '*...a joint capital...*'. Then record the next character which is 'o'. So the first simulated string is 'jo'. This process is repeated until we have simulated a string of length N. This is illustrated in the extended example

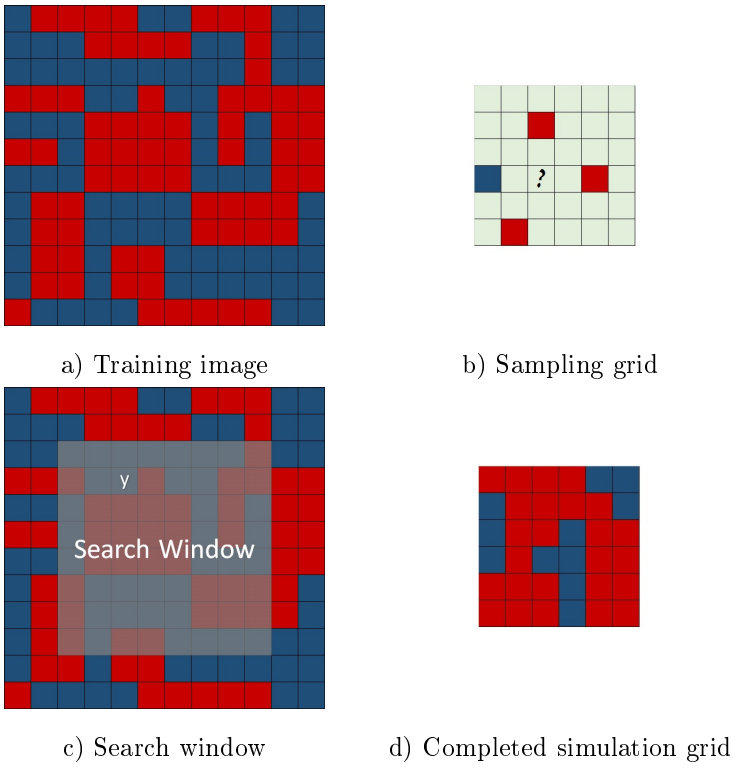
to follow. We now consider the 'The Ransom of Red Chief' text piece consisting of 100 paragraphs. If the language model is applied to this text. We then obtain the following text simulation:

'*en mine and Middle of 'em ever closed to smash him forget the Black Scout jumping him as stated. He's you got back some wide blankets and he way off his head.*'

This simulation has the same writing style and language structure as that used by the author of the original text. This is done by sampling strings directly from the original text. The generalized idea is to sample directly from some data source. This results in a simulation which represents the underlying structure of the data source. Direct Sampling aims to use this technique on spatial structures captured in a training image.

2.1.2 The Direct Sampling algorithm

According to Mariethoz, Renard and Straubhaar [32] it is common for multiple point geostatistics (MPS) algorithms to use training images. To illustrate how DS simulates from a training image consider the example in Figure 2.2. Suppose we have a spatial process represented by Figure 2.2 a). This is called the training image (TI). Also consider Figure 2.2 b) which is the grid to be completed (simulated) based on



a) Training image b) Sampling grid

c) Search window d) Completed simulation grid

Figure 2.2: Illustration of the Direct Sampling algorithm. a) The training image illustrating the spatial process, b) the simulation grid to be completed (simulated) containing the conditional data, c) the search window for the sampling of the node marked with '?', d) the completed simulation grid.

the conditional data. This is called the simulation grid (SG). In this grid the red and blue blocks are the conditional data points. The light green blocks are the empty nodes (unknown points). These colours represent the two discrete values of the nodes in this example. These nodes could also have multiple discrete values or any real values. We now illustrate the simulation of one single node. Consider the node in Figure 2.2 b) marked with a '?'. We will refer to this node as $\mathbf{x} = (4, 3)$, row 4 and column 3. The first step is to find the k nearest neighbours of $\mathbf{x}$ from the conditional data. If we let $k = 4$ then we have:

$$\mathcal{N}_4(\mathbf{x}) = \{(2, 3), (4, 1), (4, 5), (6, 2)\}.$$

Next we determine the lag vectors $\mathcal{L}_4(\mathbf{x}) = \mathcal{N}_4(\mathbf{x}) - \mathbf{x}$,

$$\begin{aligned}\mathcal{L}_4(\mathbf{x}) &= \{\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3, \mathbf{l}_4\} \\ &= \{(-2, 0), (0, -2), (0, 2), (2, -1)\}.\end{aligned}$$

The vector subtraction used to determine the lags operates by taking the difference between the row and column numbers of each neighbour and the node $\mathbf{x}$. If the difference in rows is negative/positive it indicates the neighbour is located above/below the node $\mathbf{x}$. If the difference in columns is negative/positive it indicates the neighbour is located to the left/right of the node $\mathbf{x}$. Next we find the search window in the TI. For this we use the lags. If we look at the first elements of the lags we see row-wise we have a data node 2 rows above $\mathbf{x}$, one 2 rows below, and 2 data nodes in the same row as $\mathbf{x}$. This means that the search window needs to be from the third row of the training image so that each node in the search window has at least 2 rows above it. Similarly the search window can only be up to the third last row in the TI so that each node has at least 2 rows below it. Similarly for the columns using the second elements of the lags, we see that each node in the search window needs to have at least 2 columns to its left and 2 columns to its right. The search window which satisfies these conditions is illustrated in Figure 2.2 c).

The data event of $\mathbf{x}$, which is the values of the neighbours of $\mathbf{x}$, is

$$\begin{aligned}\mathcal{D}_4(\mathbf{x}, \mathcal{L}_4(\mathbf{x})) &= \{Z(\mathbf{x} + \mathbf{l}_1), Z(\mathbf{x} + \mathbf{l}_2), Z(\mathbf{x} + \mathbf{l}_3), Z(\mathbf{x} + \mathbf{l}_4)\} \\ &= \{Z(2, 3), Z(4, 1), Z(4, 5), Z(6, 2)\} \\ &= \{1, 0, 1, 1\}\end{aligned}$$

where the $Z(x, y)$ function indicates the value of the node in row x and column y. Here 1 indicates a red node and 0 a blue node. Now to sample we search for a similar data event in the search window. We scan the search window in a random order. Consider the randomly chosen node $\mathbf{y} = (4, 5)$ marked in Figure 2.2 c) as $\mathbf{y}$. The data event of this node $\mathbf{y}$ is

$$\begin{aligned}\mathcal{D}_4(\mathbf{y}, \mathcal{L}_4(\mathbf{x})) &= \{Z(\mathbf{y} + \mathbf{l}_1), Z(\mathbf{y} + \mathbf{l}_2), Z(\mathbf{y} + \mathbf{l}_3), Z(\mathbf{y} + \mathbf{l}_4)\} \\ &= \{Z(2, 5), Z(4, 3), Z(4, 7), Z(6, 4)\} \\ &= \{1, 1, 0, 1\}.\end{aligned}$$

Using the Euclidean distance measure between the data events of nodes $\mathbf{x}$ and $\mathbf{y}$,

$$D = \sqrt{\sum \left(\mathcal{D}_4(\mathbf{x}, \mathcal{L}_4(\mathbf{x})) - \mathcal{D}_4(\mathbf{y}, \mathcal{L}_4(\mathbf{x}))\right)^2} = 1.414.$$

We now store the value of the node $Z(\mathbf{y}) = 0$ and the Euclidean distance measure $D = 1.414$. We do this only if the current node is the best match up to that point.

We can now continue this scan through the search window and hope we find a perfect neighbourhood match for $\mathbf{x}$. This will likely never happen. To address this problem we introduce an acceptance threshold. So when we find a node with distance D less than the threshold, say $t = 1$, then we end the search. This first randomly chosen node $\mathbf{y}$ is not a good enough match yet, so we move on to another randomly chosen node $\mathbf{w}$. Suppose we randomly arrive at point $\mathbf{w} = (9, 6)$ which has data event:

$$\mathcal{D}_4(\mathbf{w}, \mathcal{L}_4(\mathbf{x})) = \{1, 0, 1, 1\}.$$

As seen $\mathcal{D}_4(\mathbf{w}, \mathcal{L}_4(\mathbf{x}))$ is an exact match for $\mathcal{D}_4(\mathbf{x}, \mathcal{L}_4(\mathbf{x}))$ so $D = 0$. When we find a node $\mathbf{w}$ which has a good enough neighbourhood match we assign $Z(\mathbf{w}) = 0$ to the empty value $Z(\mathbf{x})$. This process is repeated until the simulation grid is completed. The simulation for this example is shown in Figure 2.2 d).

We summarize this algorithm in Figure 2.3. The Direct Sampling algorithm starts of with a training

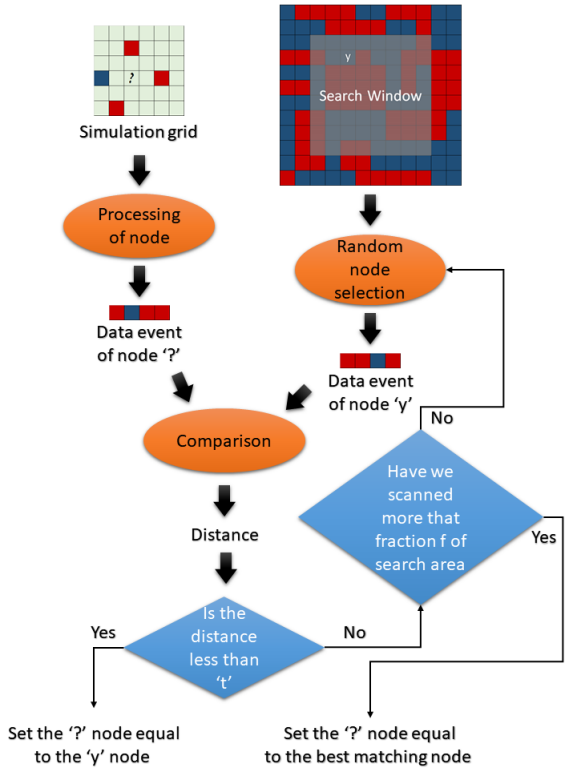


Figure 2.3: Visual Illustration of the Direct Sampling algorithm explained in Section 2.1.2.

image, a simulation grid and three tuning parameters. These parameters are the acceptance threshold t, the fraction of the search window to scan f and the maximum neighbourhood size k. The purpose of the f parameter is to balance computational time and accuracy. Naturally we will investigate every single node in the searching window until we find the best match. But this will be extremely inefficient, so for this algorithm we only scan a fraction f of the nodes in the search window. The best match out of these nodes is then assigned. The purpose of the parameter k is to restrict the size of the neighbourhoods considered. If these neighbourhoods become too large the number of options in the search window will become very limited causing accuracy to drop. Together these three parameters can be tuned to find the balance between accuracy and speed required. The steps of the Direct Sampling algorithm are as follows:

I. If there are conditioning (known/given) data points available then assign each one to the closest unknown grid node in the simulation grid (SG).

II. For each of the remaining empty nodes $\mathbf{x}$ in the SG:

   i. Find the k nearest neighbours of the node $\mathbf{x}$, $\mathcal{N}_k(\mathbf{x}) = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k\}$. If no neighbours can be found, select a random node $\mathbf{y}$ from the Training Image. Its value $Z(\mathbf{y})$ is assigned to the value of $\mathbf{x}$, $Z(\mathbf{x})$. The Z function indicates the value of the node $\mathbf{x}$. Move on to the next empty node.

   ii. Calculate the lag vectors, $\mathcal{L}_k(\mathbf{x}) = \{\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_k\}$, of $\mathcal{N}_k(\mathbf{x})$, where $\mathbf{l}_i = \mathbf{x}_i - \mathbf{x}$.

   iii. Define the data event of node $\mathbf{x}$, $\mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})) = \{Z(\mathbf{x} + \mathbf{l}_1), Z(\mathbf{x} + \mathbf{l}_2), ..., Z(\mathbf{x} + \mathbf{l}_k)\}$. This is a vector containing the values of the neighbours of $\mathbf{x}$.

   iv. Define a search area in the training image. This is an area which contains all the nodes, $\mathbf{y}$, in the training image (TI), whose neighbourhoods $\mathcal{N}_k(\mathbf{y}) = \mathbf{y} + \mathcal{L}_k(\mathbf{x})$ are also located in the training image.

   v. Starting at a random node $\mathbf{y}$, for each randomly selected $\mathbf{y}$:

      a. Find the data event of $\mathbf{y}$, $\mathcal{D}_k(\mathbf{y}, \mathcal{L}_k(\mathbf{x}))$.

      b. Calculate the distance between the data events of $\mathbf{x}$ and $\mathbf{y}$, $D(\mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})), \mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})))$.

      c. Store $Z(\mathbf{y})$ and $D(\mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})), \mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})))$ if it has the smallest distance for the current search.

      d. If $D(\mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x})), \mathcal{D}_k(\mathbf{x}, \mathcal{L}_k(\mathbf{x}))) \leq t$ then assign $Z(\mathbf{y})$ to the value of node $\mathbf{x}$, $Z(\mathbf{x})$. Move on to the next empty node.

      e. If the number of iterations of steps $a - d$ is more than $f \times K$, where K is the number of nodes in the search window, then assign the $Z(\mathbf{y})$ with the smallest distance to the value of node $\mathbf{x}$, $Z(\mathbf{x})$. Move on to the next empty node.

2.2 Bunch-pasting Direct Sampling

Direct Sampling was developed with the aim of being less computationally intensive than algorithms like SNESIM and ENESIM [32]. This is achieved to an extent, but unfortunately it is not fast enough. An attempt to improve the speed of DS was made by Razaee et al [38] when they developed bunch-pasting DS. As described in Section 2.1.2 Direct Sampling finds a node in the training image with a neighbourhood similar to a empty node in the SG. It then simply 'pastes' the training image node into the simulation grid. Thus there is no distribution estimation involved in this process. Hence it is proposed in [38] to simply paste a bunch of nodes from the training image simultaneously. As seen in Figure 2.4 when the single node method finds a neighbourhood match it pastes a single node. Bunch-pasting on the contrary pastes all the nodes in the region corresponding to empty nodes with that neighbourhood. Razaee together with the original DS author, Mariethoz, argue that this new method will yield similar results to the single node pasting Direct Sampling algorithm [38]. This is because each single node in the bunch of nodes being pasted will have a similar neighbourhood to the node it fills. These nodes may not be the most optimal matches for the nodes they fill. They will however be good enough to get good performance whilst gaining a significant increase in computational efficiency.

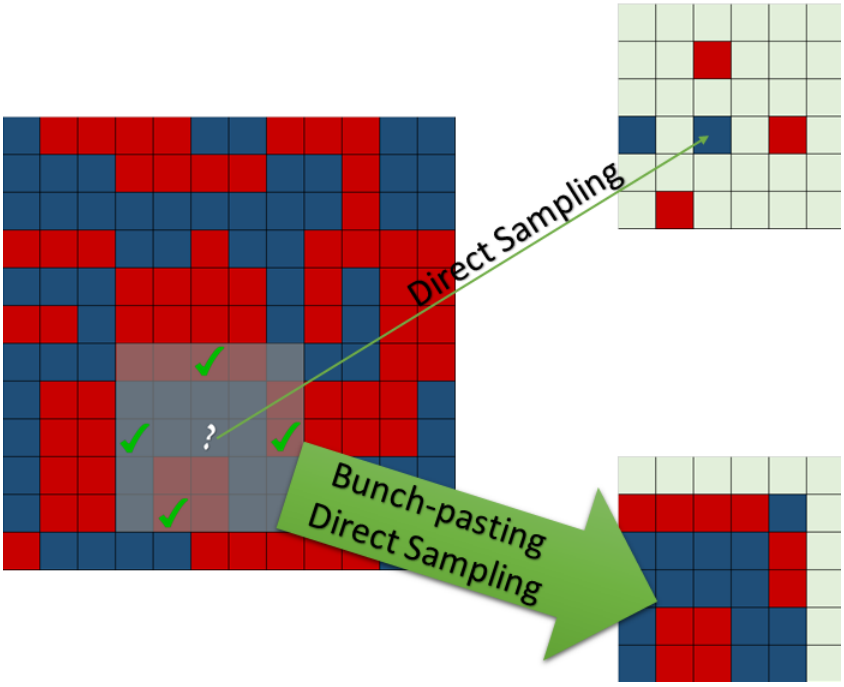


Figure 2.4: Direct Sampling vs Bunch-pasting Direct Sampling (BDS).

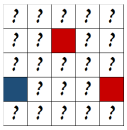
The advantages of using the Bunch-pasting Direct Sampling method are:

- Multiple nodes are simulated at once resulting in reduced computation time.
- The shape of the neighbourhood filter does not have to be a square. A disk or any other shape filter

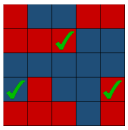
desired can be used to improve results.

- A fixed set of windows of fixed shape and size are scanned at each iteration. This eliminates the need for finding a new search area for each iteration. This also reduces computation time.

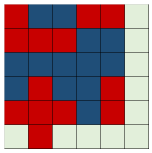
Consider again the training image and simulation grid in Figure 2.2 a) and b) respectively. Also consider a square shape filter of size 5×5. In Figure 2.5 a) is the first window from the simulation



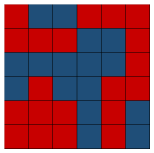
a) First SG window to fill.



b) First randomly selected window from the TI.



c) Sampling grid after first window simulated.



d) Completed simulation grid.

Figure 2.5: Illustration of the Bunch-pasting Direct Sampling algorithm. a) The first simulation grid window to be completed, b) the first randomly selected window from the training image, c) the simulation grid after pasting the first bunch of nodes from the training image, d) the completed simulation grid.

grid to be completed. To complete this window we need to find a window in the training image with similar values at the known nodes. Consider the randomly selected window in Figure 2.5 b) from the TI. The green ticks in this window shows that it is a perfect match in terms of known data nodes to the window in a). If it was not a perfect match we would calculate the distance and compare it to a acceptance threshold as done in Section 2.1.2. Since we have found a good enough match for this window all the nodes are now pasted from the training image window into the simulation grid. It is important to note that conditional data or previously simulated nodes will not be overwritten. In Figure 2.5 d) is the completed simulation grid obtained using the Bunch-pasting DS. This simulation was obtained using just 4 searches in the training image, compared to the 22 searches used by Direct Sampling. This simulation process is illustrated visually in Figure 2.6.

The Bunch-pasting DS has the same input as DS, except for the neighbourhood size. Instead of specifying a maximum neighbourhood size, for this algorithm a set neighbourhood filter is defined. This filter has a set shape as size. For this algorithm we define some new notation. $\mathbf{X}$ refers to a window of nodes in the simulation grid which is selected by the neighbourhood filter. $\mathbf{Y}$ refers to a window of nodes in the training image which is selected by the neighbourhood filter. The steps of the Bunch-pasting Direct Sampling algorithm are summarized as follows:

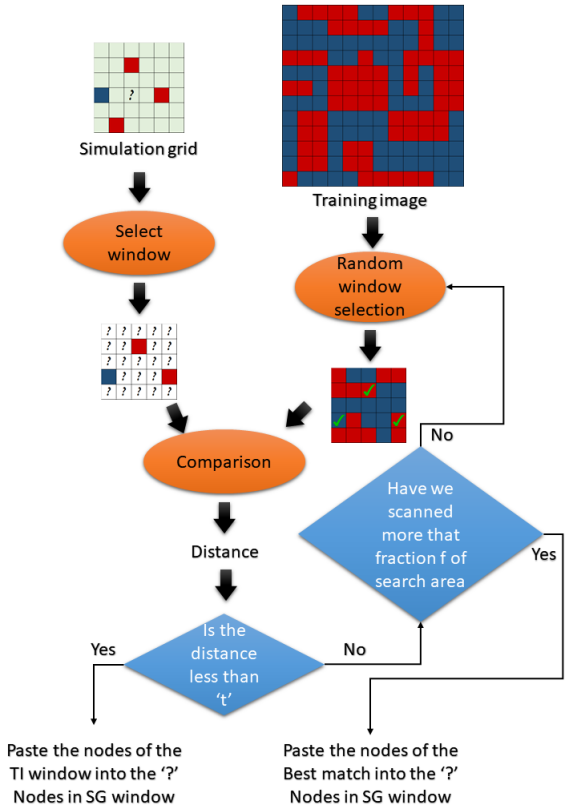


Figure 2.6: Visual Illustration of the Bunch-pasting Direct Sampling algorithm explained in Section 2.2.

I. If there are conditioning (known/given) data points available then one is assigned to the closest unknown grid node in the simulation grid.

II. Define the shape and size of the neighbourhood filter F to be used.

III. For each window $\mathbf{X}$, with the same shape and size as F, in the simulation grid (SG). If there are empty nodes in $\mathbf{X}$:

   i. Starting at a random window $\mathbf{Y}$, with the same shape and size as F, search through the TI and:

      a. Calculate the distance D_{XY} between the known neighbourhood windows $\mathbf{X}$ and $\mathbf{Y}$. This multivariate distance is the Euclidean distance between the known nodes in $\mathbf{X}$ and their counterparts in $\mathbf{Y}$.

      b. Store $\mathbf{Y}$ and D_{XY} if D_{XY} is the smallest distance for the current search.

      c. If $D_{XY} \leq t$ then paste (assign) the nodes in $\mathbf{Y}$ corresponding to the empty nodes in $\mathbf{X}$ into these positions. Move on to the next window.

      d. If the number of iterations of steps $a - c$ is more than $f \times K$, where K is the number of windows in the training image, then paste (assign) the nodes in the window $\mathbf{Y}$ corresponding

to the empty nodes in $\mathbf{X}$ into these positions, where $\mathbf{Y}$ is the best match for the current search. Move on to the next window.

2.3 Feature matching using Direct Sampling

We have seen that it is possible to simulate an image using Bunch-pasting Direct Sampling (BDS). We argue that it is also possible to use BDS as a feature matching algorithm. To start off consider again the example used in Figure 2.5. In Figure 2.7 we show how the Bunch-pasting Direct Sampling algorithm creates links between the simulation grid and the training image. We propose a novel feature matching

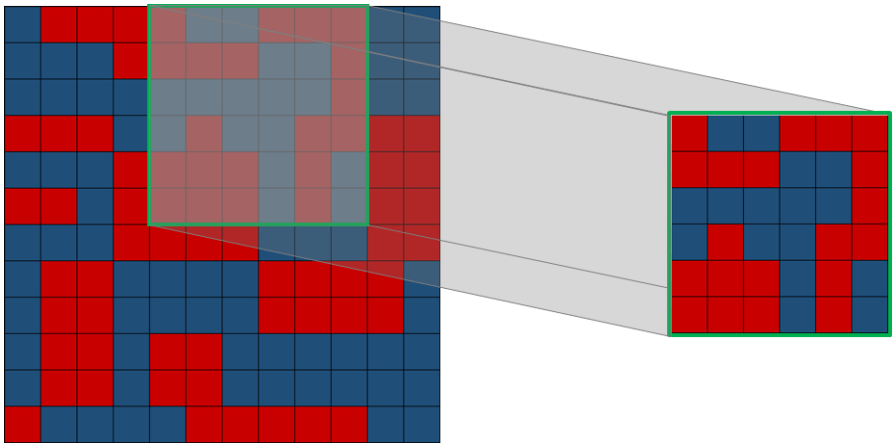


Figure 2.7: Illustration of Bunch-pasting DS pasting a region whilst simulating. During the process the locations of the nodes in the window being pasted and the nodes being completed can be stored. This creates links between the training image and simulation grid.

algorithm using these links created. Feature matching or localisation is the process of matching regions in one image to similar regions in another. Let us consider two images, let the first image be the 'training image' and the second the 'simulation grid'. As mentioned here simulating the simulation grid using the BDS will then create links between these two images. A full image does however not naturally make a 'simulation grid'. To convert an image to a simulation grid we need to select pixels in the image to be empty nodes. This allows BDS to link features in the simulation grid to features in the training image. We call this new spatial feature matching algorithm Direct Sampling Feature Matching (DSFM). In Figure 2.8 we visually illustrate this new feature matching method. As seen the first image is assigned as the 'training image' and the second image as the 'simulation grid'. Randomly selected nodes in the 'simulation grid' are then set as empty. Bunch-pasting Direct Sampling then creates links between the two images. These links are interpreted as matched features.

We now need to decide on a method for selecting empty nodes in the 'sampling grid'. A very simple starting point is to just select random nodes to be empty. The input to this method includes two images.

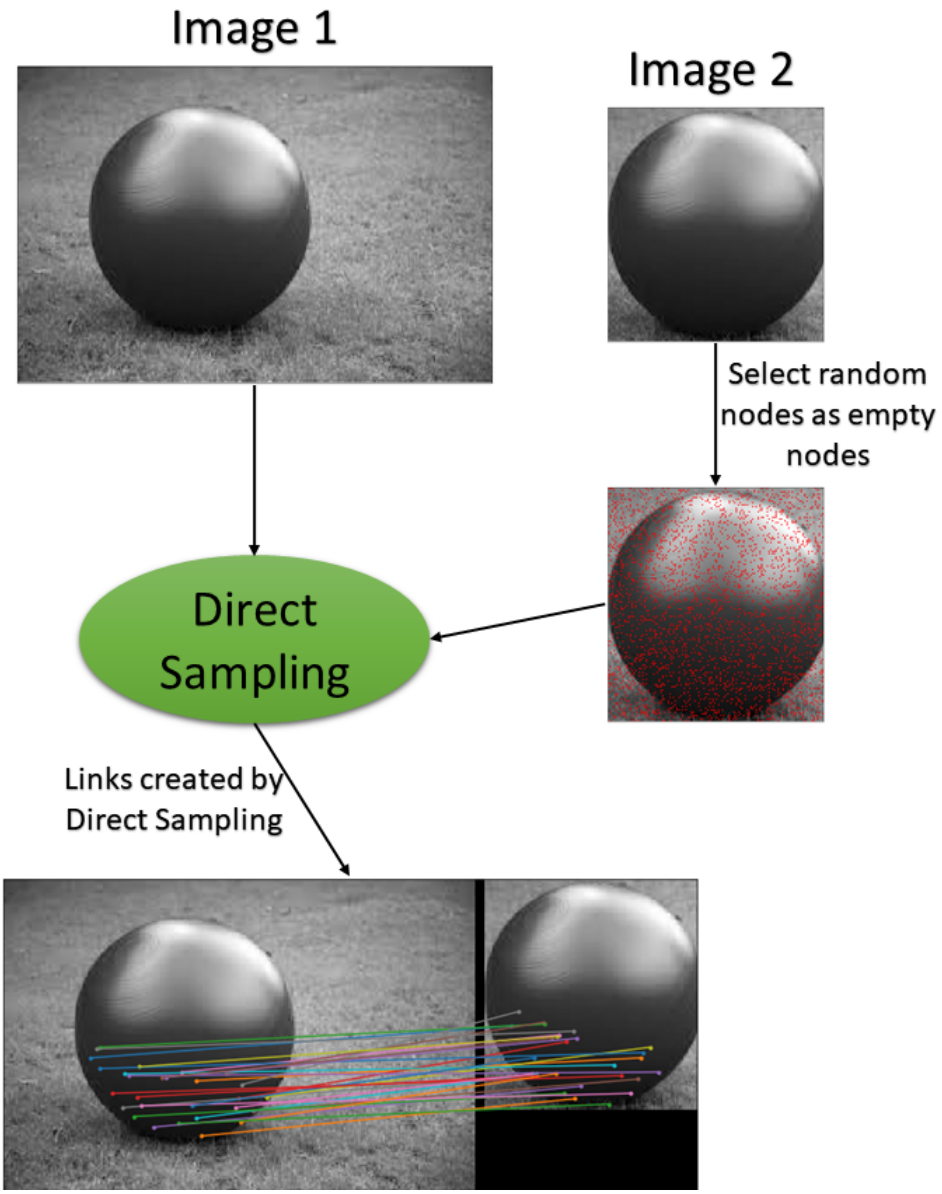


Figure 2.8: Visual illustration of the Direct Sampling Feature Matching algorithm.

The algorithm aims to match the features in the second image to similar features in the first. The algorithm also contains 5 tuning parameters. Firstly a parameter indicating how many random empty nodes to create in the simulation grid. This is the α parameter, and αN randomly chosen nodes will be set as empty, where N is the total number of pixels in the second image. We also have the three BDS algorithm parameters as described in Section 2.2. We then select the strongest links to match features. For this we have a β parameter which selects the strongest $\beta\alpha N$ links. The strength of a link is measured using the distance between the neighbourhood of the node in the first image and the neighbourhood of the node in the second image. The steps of the Direct Sampling Feature Matching algorithm follows:

I Set the first image as the 'training image'.

II Select αN random nodes in the second image. Set these nodes as empty.

III Set the second image with its empty nodes as the 'simulation grid'.

IV Perform Bunch-pasting Direct Sampling.

V Select the strongest $\beta\alpha N$ links created by the BDS algorithm between the two images. These links show the most pertinent features matched by the DSFM algorithm.

VI Determine the distance between the two images. This distance is the distance related to the weakest link created. Hence the distance between two images is

$$\max_{l \in \mathcal{M}} \mathcal{D}(l),$$

where $\mathcal{M}$ is the set of links created between two images and $\mathcal{D}(l)$ is the distance between the neighbourhoods of the pixels linked by link l.

2.4 Concluding remarks

In this chapter we have given the background of the Direct Sampling algorithms. We have also shown that Bunch-pasting Direct Sampling, a computationally more efficient version of DS, can be used as a feature matching tool. We will next introduce a more advanced feature detector which could potentially improve this technique further.

Chapter 3

Feature detection using the Discrete Pulse Transform

In Chapter 1 we discussed multiscale feature detection techniques such as SIFT and SURF. We also discussed the possibility of using the Discrete Pulse Transform (DPT) as a feature detector. Here we discuss the DPT in detail. We also discuss a graphical DPT implementation called the Roadmaker's Pavage, and then we discuss a graphical method for feature detection using the Roadmaker's Pavage algorithm.

3.1 Introduction

Digital signals refer to a discrete set of data points constructed from a continuous analogue process. A simple example of this is the market value of a stock over time. The true process is a continuous process but the digital signal from this process is market values measures, say, every day. A spatial example of digital signals is a digital image, where the discrete red, green and blue colour intensities are measured as vectors at discrete points called pixels. The true image however represents an infinite number of colours on a continuous plane. The Discrete Pulse Transform (DPT) is a method for processing signals at multiple scales.

The original development of the Discrete Pulse Transform (DPT) as a 1-dimensional multiscale analysis tool was done by Carl Rohwer and co-authors in multiple papers. The most significant results and theory appears in a book published in 2005 by Rohwer [42]. The development of this theory is also discussed by Rohwer in [39, 40, 43, 44, 41, 42, 45]. The DPT decomposes a signal into a collection of pulses over all possible discrete scales, these pulses are signals made up of a constant and zeros. The original signal can be reconstructed by summing up the collection of pulses.

Research into the multiscale decomposition of a signal originated when in 1822 French mathematician

Joseph Fourier showed that some functions can be written as an infinite sum of harmonics in his book Analytical Theory of Heat [14]. The Discrete Fourier Transform (DFT) arose from this research by Fourier. However an issue which arises with this algorithm is that it is weak at handling discontinuities in the data [43]. An improvement to the DFT is the Wavelet Transform. The Discrete Wavelet Transform (DWT) is widely used as a signal decomposition method, and literature such as [15] discusses the use of this method for image processing.

In other developments the median transform algorithm was developed. It is a non-linear smoother which gives it the ability to preserve edges. According to [43] some of the disadvantages of the median transform is that it is a computationally expensive algorithm and no theoretical background predicting its behaviour exists.

It has been theoretically proven that the DPT developed by Rohwer has many desirable properties. In [43] it is proven that the DPT in 1D is a consistent separator of data sequences. The DPT is obtained through the so called LULU smoothers L_n and U_n. In [5] the 1D version of the DPT is extended to a multidimensional Discrete Pulse Transform and LULU operators, also extending the properties into a multidimensional environment. This opens up numerous fields of application for the DPT. The one that we will be exploring here is significant feature detection, specifically for identifying textures. The extension of these operators to multiple dimensions relies on the use of a morphological connection developed by Serra [48].

Definition 3.1.1. Morphological Connectivity [13]
Let X be an arbitrary non empty set. A family Z of subsets of X is called a connected class if:

i) $\phi \in$ Z

ii) $\{x\} \in$ Z for each $x \in B$

iii) For any family $\{Z_i\} \subseteq$ Z we have

$$\underset{i\in I}{\cap} Z_i \neq \phi \Rightarrow \underset{i\in I}{\cup} Z_i \in \text{Z}$$

If a set C belongs to a connection Z then C is called a connected set.

Let the set of all connected sets, V, of size $n+1$ including the point x be denoted by $\mathcal{N}_n(\mathbf{x})$. Then for a multidimensional signal $f(\mathbf{x})$, with $\mathbf{x}$ the domain over which the signal f is defined, the LULU operators [5] are defined as:

$$L_n(f)(\mathbf{x}) = \max_{V\epsilon\mathcal{N}_n(\mathbf{x})} \min_{y\epsilon V} f(y),$$
$$U_n(f)(\mathbf{x}) = \min_{V\epsilon\mathcal{N}_n(\mathbf{x})} \max_{y\epsilon V} f(y).$$

The Discrete Pulse Transform of a signal $f(\mathbf{x})$ is then determined as follows:

$$DPT(f)(\mathbf{x}) = \{D_1(f)(\mathbf{x}), D_2(f)(\mathbf{x}), ..., D_N(f)(\mathbf{x})\}$$

where

$$D_1(f)(\mathbf{x}) = (I - P_1)f(\mathbf{x})$$

and

$$D_n(f)(\mathbf{x}) = (I - P_n) \circ Q_{n-1}f(\mathbf{x})$$

with $Q_{n-1} = P_1 \circ P_2 \circ ... \circ P_{n-1}$, so $Q_{n-1}f(\mathbf{x})$ is the smoothed sequence after $n-1$ applications of the smoothing operator P_i, and $D_n(f)(\mathbf{x})$ is the signal containing all the pulses of scale n derived from the signal $f(\mathbf{x})$. Here $P_n = L_n \circ U_n$ or $U_n \circ L_n$.
The signal can then be represented by the sum of a collection of pulses $\{p_{ni}(f)\}$ as follows:

$$f(\mathbf{x}) = \sum_{n=1}^{N} D_n(f)(\mathbf{x}) = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} p_{ni}(f)(\mathbf{x}).$$

Each $D_n(f)$ is a scale layer of scale n made up of all the pulses of that scale, $\{p_{n1}(f), p_{n2}(f), ..., p_{n\gamma(n)}(f)\}$, where $\gamma(n)$ is the number of pulses of scale n.

To illustrate the DPT decomposition in 1D consider the definitions of L_n and U_n in 1D for the signal $f(\mathbf{x})$ [5],

$$(L_n(f)(x_i)) = \max\{\min\{f(x_{i-n}), ..., f(x_i)\}, ..., \min\{f(x_i), ..., f(x_{i+n})\}\},$$

$$(U_n(f)(x_i)) = \min\{\max\{f(x_{i-n}), ..., f(x_i)\}, ..., \max\{f(x_i), ..., f(x_{i+n}\}\}.$$

In Figure 3.1 the DPT decomposition of the small signal $f(\mathbf{x}) = \{7, 3, 5, 5, 2\}$ is shown. For this example we obtain the decomposition $\{D_1(f), D_2(f), D_3(f), D_4(f), D_5(f)\}$ with:

$$D_1(f)(\mathbf{x}) = \{4, 0, 0, 0, 0\},$$

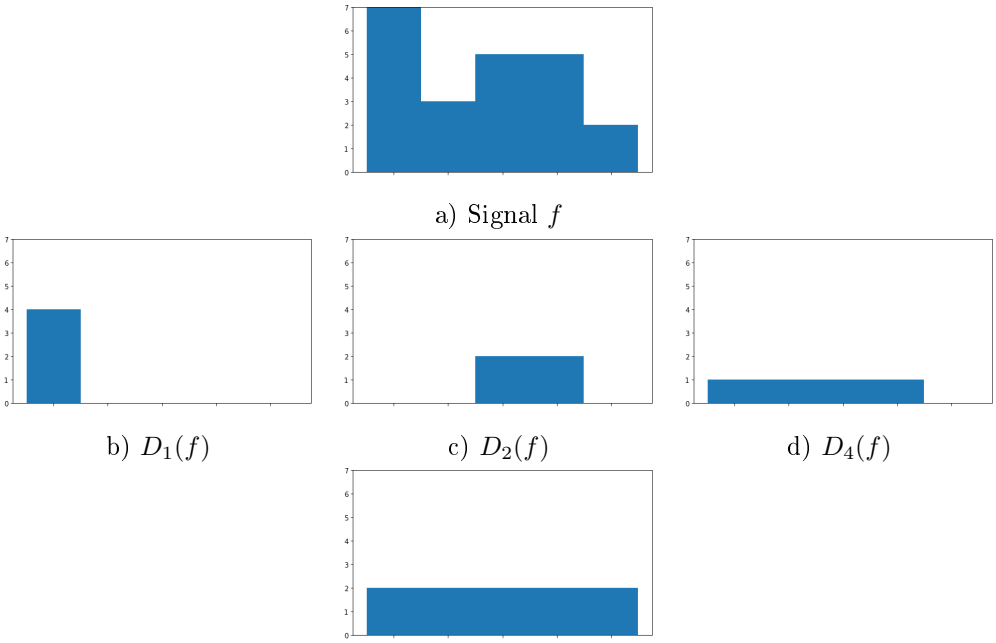
$$D_2(f)(\mathbf{x}) = \{0, 0, 2, 2, 0\},$$

$$D_3(f)(\mathbf{x}) = \{0, 0, 0, 0, 0\},$$

$$D_4(f)(\mathbf{x}) = \{1, 1, 1, 1, 0\},$$

$$D_5(f)(\mathbf{x}) = \{2, 2, 2, 2, 2\}.$$

To explain exactly how the results in Figure 3.1 were obtained we need to understand what a local minimum or maximum set is. A local minimum(maximum) set is a connected set of points in a signal which all have values less(greater) than their neighbours [5]. Now to decompose the signal we start by applying the U_1 operator, applying this operator removes all the local maximum sets of scale 1. When removing a set we remove the part of the set which is greater or less than its nearest neighbour, in terms of distance. We then store this 'piece' we removed as a pulse of scale 1. Next we remove all local minimum sets of scale 1 from the smoothed signal. We repeat this process for scale $= 2, 3, ...$ As we can see in Figure 3.1 the first pulse removed is the local maximum set of scale 1 at the first position from the left.



a) Signal f

b) $D_1(f)$ c) $D_2(f)$ d) $D_4(f)$

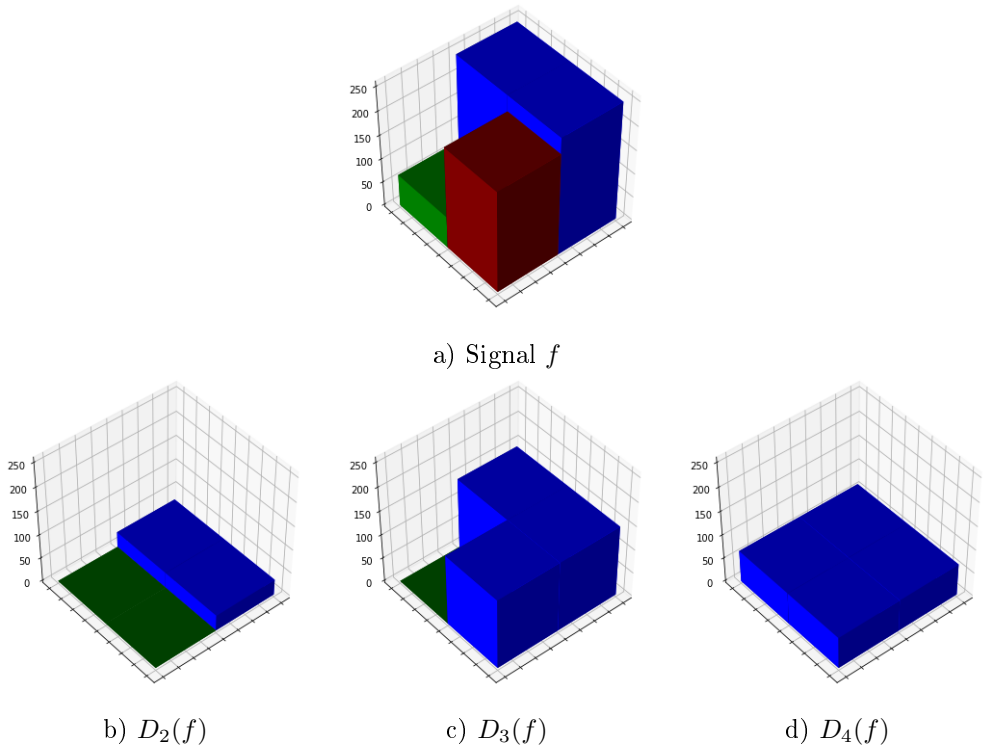
a) $D_5(f)$

Figure 3.1: DPT decomposition of the signal $f(\mathbf{x}) = \{7, 3, 5, 5, 2\}$. The scale layers are illustrated as b) scale 1, c) scale 2, d) scale 4 and e) scale 5. Scale layer 3 is not illustrated because it is just a signal of 0's.

This results in the smoothed signal $\{3, 3, 5, 5, 2\}$. We see that there are no local minimum sets of scale 1 to remove. Note that the point of height 2 on the right is not a local minimum set because all edge points are connected to the infinite scale 0 signal. Next U_2 is applied and removes the local maximum set of scale 2 $\{0, 0, 2, 2, 0\}$. This process is repeated to arrive at the results presented in Figure 3.1.
To illustrate the process in 2D, consider the 2×2 image in Figure 3.2 a). In Figure 3.2 it is illustrated how a 2D signal such as an image is decomposed, the decomposition is done by the Roadmaker's Pavage algorithm [52]. In Figure 3.2 a) we see that there are no local extrema of scale 1 in the signal. Again this is because all edge nodes are connected to a infinite scale, 0 signal. As seen in b) there is a local maximum set of scale 2, this is the blue nodes in the signal shown in a). When the pulse in b) is removed it results in a signal where the blue nodes in a) have the same value as the maroon node in a). We then find the pulses of scales 3 and 4 in a similar manner. This results in the decomposition seen in Figure 3.2.

3.2 Implementation algorithms for the Discrete Pulse Transform

One of the initial challenges of applying the DPT with LULU operators was that it is computationally expensive. When the DPT is applied to a 2D signal, such as an image, a connectivity between points in the signal needs to be defined. For example when using 4-connectivity there are four neighbours to consider for every point in the signal. Hence a very large number of points should potentially be scanned



a) Signal f

b) $D_2(f)$ c) $D_3(f)$ d) $D_4(f)$

Figure 3.2: Pulse decomposition of $2D$ signal. b) The local maximum set pulse of scale 2, c) the local maximum set pulse of scale 3 and d) the pulse of size 4. There are no pulses of scale 1 and hence it is not illustrated.

during the process, which will be very computationally expensive. Different connectivity functions are illustrated in Figure 3.3. In Figure 3.4 it is illustrated how fast the number of connected sets grow as the neighbourhood size grows. We see that for $n = 1$ we have 4 connected sets and for $n = 2$ we have 12 connected sets. For 4-connectivity each points has 4 neighbours, and a connected set of 2 points has 6 neighbours. 8-connectivity results in 8 neighbours for a single point and 10 neighbours for a set, and similarly 12-connectivity results in 12 neighbours for a single point and 16 neighbours for a set. This illustrates how the size of the neighbourhood sets that need to be investigated to find local minimums and maximums can become very large. Considering a signal like a image with tens of thousands of pixels, this shows that the DPT decomposition can be very computationally intensive. This problem has since been relieved by the development of algorithms such as the Roadmaker's algorithm in 1D by Laurie [22] and the more efficient 2D DPT in [13]. A generalized Roadmaker's algorithm for d dimensions is also introduced in [21]. Further development was made by Stoltz [52] when he developed a graphical implementation called the Roadmaker's Pavage. In this research the algorithm which will be used for DPT signal decomposition is the Roadmaker's Pavage algorithm, due to its efficient storage and decomposition in a graph setting.

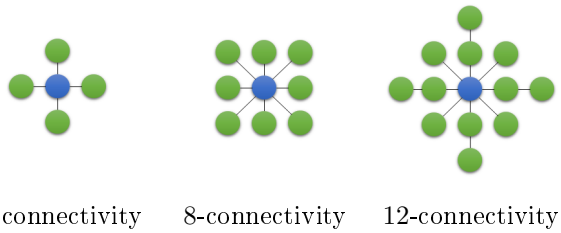


Figure 3.3: Different connected neighbourhood sizes.

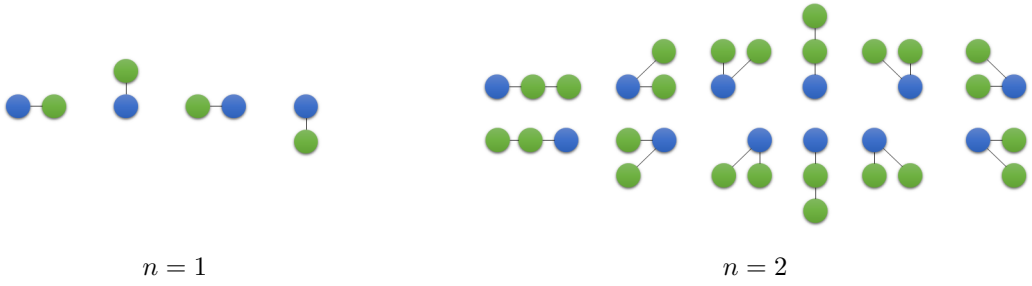


Figure 3.4: All possible connected sets V of size n included in $\mathcal{N}_n(\mathbf{x})$, for $n = 1$ and $n = 2$ only. The blue dot is the central node $\mathbf{x}$ and the green dots the neighbouring nodes.

3.2.1 Roadmaker's algorithm

Imagine smoothing a dirt road. This could be done by simply filling up all pits and flattening all bumps. This logic was followed by Laurie [22] when developing the original 1D Roadmaker's algorithm. Picture the signal shown in Figure 3.1 as the side view if a dirt road. This algorithm starts by removing all bumps (local maximum sets) of scale 1 from the road. The signal seen in Figure 3.1 can be seen as the bump removed in this step. The algorithm then removes all pits in the road (local minimum sets). We assume there is a flat road of height 0 before and after the signal 'road'. Next bumps of scale 2 are removed, the resulting bumps can be seen in Figure 3.1. This process is repeated for all scales to obtain the same results as previously obtained in Figure 3.1. A improved implementation of the DPT in 2D in terms of memory-efficiency was developed by Fabris-Rotelli and van der Walt [13]. A generalized multiple dimension Roadmaker's algorithm was also developed by Laurie [21]. We do not go into further detail regarding these algorithms as the Roadmaker's Pavage algorithm provides the most efficient storage and access to pulses, as well as the ability to represent a signal of any dimension.

3.2.2 Roadmaker's Pavage algorithm

The Roadmaker's Pavage Algorithm, developed by Stoltz [52] is a graph based DPT algorithm. This algorithm is an alternative approach to the one followed by Laurie in the Roadmaker's algorithm [22, 21], in the sense of the data structure used. To illustrate how this algorithm decomposes a signal, we consider

the simple 2×2 signal from Figure 3.2 a), this signal is represented as a grey-scale image in Figure 3.5.

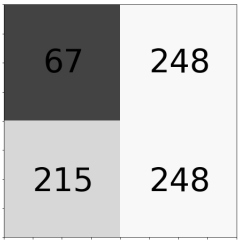
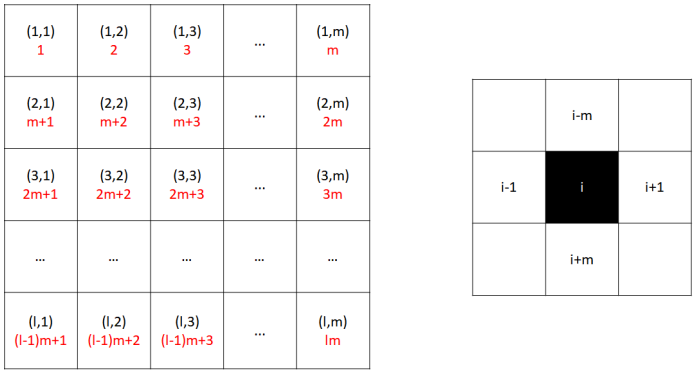


Figure 3.5: Signal from Figure 3.2 a) represented as a simple grey-scale image.

We introduce the the following notation for the Roadmakers Pavage algorithm.

- $C_k(i)$ - The connectivity functions used to represent the multidimensional relationships in the data.
- W_G - The working graph is the graph which consists of nodes representing the data sequence, and edges representing the connectivity.
- P_G - The pulse graph is the graph which consists of nodes representing the pulses extracted by DPT, and arcs representing the links between pulses.
- $V_{G,i}$ - Node number i in graph G. A node has two attributes, the value and scale of the node. A node can represent a cluster of data points, where the value of that node is the data value of that cluster and the scale of the node is the number of data points in the cluster.

Step 1: Defining the connectivity functions.



a) b)

Figure 3.6: Illustration of index changes when flattening $2D$ signal. a) The difference between the indices of the $2D$ grid on which a $2D$ signal can be defined on, and the indices of the flattened $2D$ signal points. b) The 4-connectivity relation of a $2D$ signal can be represented using the indices of the flattened signal.

The connectivity functions, $C_k(i)$, are the functions used to represent the relationships between data points. These functions allow us to represent multidimensional relationships in 1D, this is done by 'flattening' a multidimensional signal into 1D and then representing the multidimensional connections like the ones show in Figure 3.6 b) using the 1D indices. This representation of multidimensional signals in 1D is visually explained in Figure 3.6. The general connectivity functions for 4-connectivity for a $l \times m$ image are:

$$C_1(i) = \begin{cases} 0 & \text{if } i = 1, m+1, 2m+1, ..., m(l-1)+1 \\ i-1 & \text{otherwise} \end{cases}$$

where $C_1(i)$ represents the pixel to the left,

$$C_2(i) = \begin{cases} 0 & \text{if } i = m, 2m, ..., lm \\ i+1 & \text{otherwise} \end{cases}$$

where $C_2(i)$ represents the pixel to the right,

$$C_3(i) = \begin{cases} 0 & \text{if } i = 1, 2, ..., m \\ i-m & \text{otherwise} \end{cases}$$

where $C_3(i)$ represents the pixel above, and

$$C_4(i) = \begin{cases} 0 & \text{if } i = (l-1)m+1, (l-1)m+2, ..., (l-1)m+(m-1), lm \\ i+m & \text{otherwise} \end{cases}$$

where $C_4(i)$ represents the pixel below.

From the DPT theory if a point is a boundary point then its neighbour added should be the infinite 0 signal. As seen from the C functions above and from Figure 3.6 a) these functions link the boundary points to the infinite 0 signal, which has index 0.

So the connectivity functions for this example are:

$$C_1(i) = \begin{cases} 0 & \text{if } i = 1, 3 \\ i-1 & \text{otherwise} \end{cases}$$

$$C_2(i) = \begin{cases} 0 & \text{if } i = 2, 4 \\ i+1 & \text{otherwise} \end{cases}$$

$$C_3(i) = \begin{cases} 0 & \text{if } i = 1, 2 \\ i-m & \text{otherwise} \end{cases}$$

$$C_4(i) = \begin{cases} 0 & \text{if } i = 3, 4 \\ i+m & \text{otherwise} \end{cases}$$

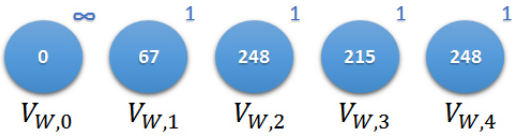
Step 2: Initialisation of graphs.

During the explanation of this section we have two columns, on the left the step is explained in words and on the right the implication of the explanation from the left on the graph is visually illustrated.

The Working graph W_G

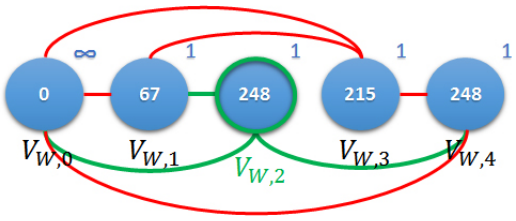
Create the zero node $V_{W,0}$ which has value 0 and scale ∞. In DPT theory it is necessary for all boundary elements of the signal to be connected to a 0 element for the domain outside the signal domain and in turn which are connected to endless 0 elements. For this reason the boundary elements of the working graph are connected a 0 node of scale ∞.

A node $V_{W,i}$ is then created for each element in the signal. These nodes have value equal to the signal value $f(x_i)$ and scale 1.



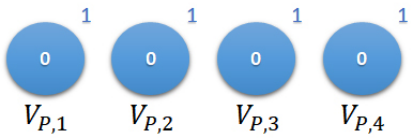
Using the connectivity functions $C_k(i)$, $k = 1, 2, ..., p$ with p being the number of connectivity functions required to define the full connectivity pattern the user wishes to use, the neighbouring nodes of a node $V_{W,i}$, $\{V_{W,C_1(i)}, V_{W,C_2(i)}, ..., V_{W,C_p(i)}\}$ are obtained. Edges are created between $V_{W,i}$ and all its neighbours. For example, the $V_{W,2}$ node has neighbouring nodes $\{V_{W,1}, V_{W,4}, V_{W,0}\}$, and the edges between $V_{W,2}$ and its neighbours are highlighted in green on the graph to the right.

$V_{W,3}$ is not a neighbour of $V_{W,2}$ since it is the node diagonally to the right and top of $V_{W,2}$ and we are using 4 connectivity which only uses the neighbours directly below, above and to the right and left.



The Pulse graph P_G

A node $V_{P,i}$ is created for each data element with value 0 and scale 1.



A virtual edge is then created between each node in the working graph $V_{W,i}$ and its corresponding node in the pulse graph $V_{P,i}$ for $i = 1, 2, .., lm$. These steps produces the initial graphs presented in Figure 3.7 a).

Step 3: Constructing the feature table.

For every node $V_{W,i}$ with $i = 1, 2, .., lm$ in W_G, if any of the neighbours of $V_{W,i}$ have the same value as $V_{W,i}$, say $V_{W,j}$, then $V_{W,j}$ inherits all the edges and virtual edges of $V_{W,i}$ and

$$Scale\,(V_{W,j}) = Scale\,(V_{W,j}) + Scale\,(V_{W,i})$$

$V_{W,i}$ is also removed from the W_G. We see that the $V_{W,2}$ node has a neighbour with equal value to itself. The above process is applied to the $V_{W,2}$ node and the resulting graphs are presented in Figure 3.7 b). Hence the initial feature table for this example consists of: $\{V_{W,1}, V_{W,3}, V_{W,4}\}$

Step 4: The decomposition.

The process is started with a scale parameter of $n = 1$ just as the LULU decomposition would start with operators L_1 and U_1. Then for all nodes, $V_{W,i}$, in the feature table of scale equal to the current scale value, sequentially set $CNode = V_{W,i}$. Before we continue this process we will define a min and max feature. A min or max feature is a local extrema or in other words a node which is a minimum or maximum in its local set. These nodes are the bumps and pits described in the Roadmaker's algorithm.

i. If $Value\,(CNode)$, this $Value()$ function extracts the value of a node from the working graph, is equal to the value of any of its neighbours, say $V_{W,j}$, then $V_{W,j}$ inherits all the edges and virtual edges of $CNode$ and

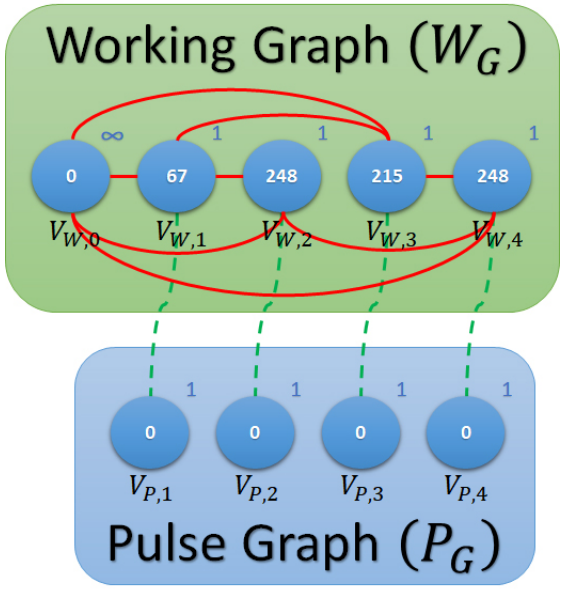
$$Scale\,(V_{W,j}) = Scale\,(V_{W,j}) + Scale\,(CNode)$$

Then $CNode$ is deleted from the W_G and $V_{W,j}$ added to the feature table.

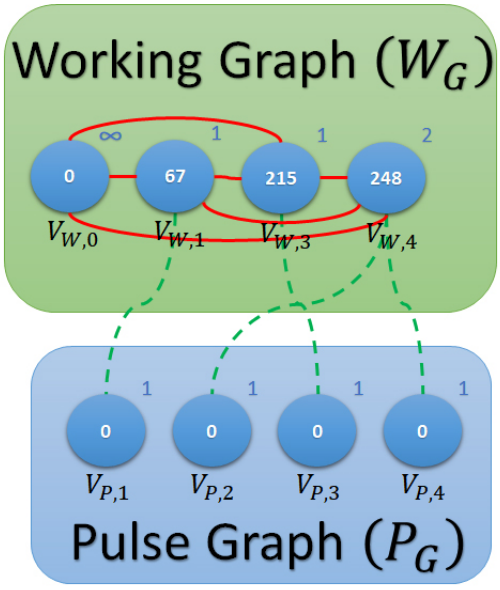
ii. If $CNode$ is a max feature then it is a pulse.

iii. If $CNode$ is a min feature, and a max feature of scale equal n is not present in the feature table then $CNode$ is a pulse.

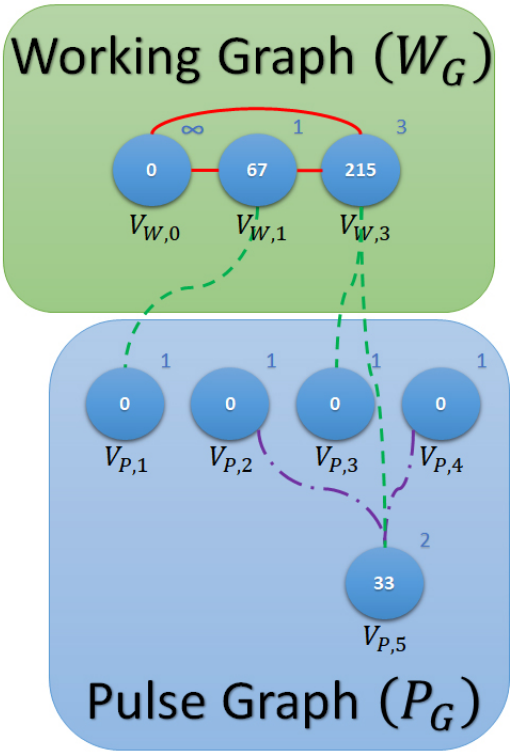
iv. If $CNode$ is a min feature, and a max feature of scale equal n is present in the feature table then $CNode$ is not a pulse.



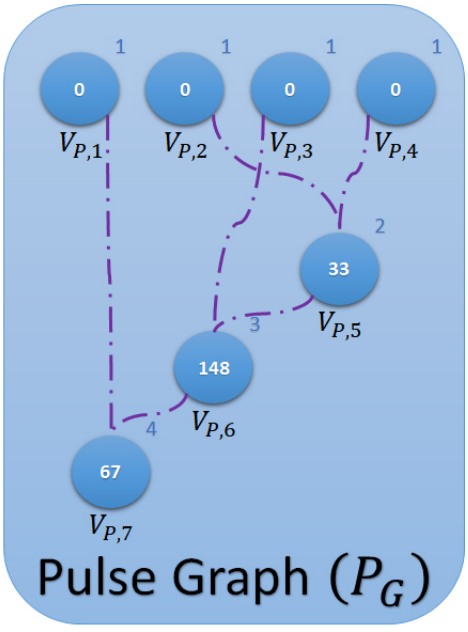
a) Initial Graphs



b)Graphs after combining equal valued nodes



c) Graphs after extracting first pulse



d) Final pulse graph

Figure 3.7: Roadmaker's Pavage graphs. From a) - d) we see the progression of the Roadmaker's Pavage graphs during the signal decomposition process.

v. If *CNode* is not a pulse nor a min feature it is removed from the feature table.

These steps perform the same task as applying the U_n filter first and then the L_n filter. This is because all the local maximums are flagged as pulses and flattened first after which all the local minimums are flagged as pulses, hence why a min feature is only a pulse once there is no max features left, and flattened. This can be swapped around to perform L_n and then U_n, but the results are similar enough for this not to be necessary. Now consider the first node of scale 1 in the feature table which is $V_{W,1}$. Set $CNode = V_{W,1}$. It can be seen on the graph above that $V_{W,1}$ does not a meet any of the criteria since it has a neighbour larger and a neighbour smaller in value so it is neither a min nor a max feature. Hence it is not a pulse and is removed from the feature table. The feature table now consists of $\{V_{W,3}, V_{W,4}\}$. Similarly $V_{W,3}$ does not meet any of the criteria so it is also removed from the feature table. Hence the feature table now consists of only $V_{W,4}$. But this remaining node in the Feature table is not of scale $n = 1$. If there are no nodes in the feature table with scale n then the current scale parameter n is increased by 1. So the current scale parameter value is increased to $n = 2$. Now set $CNode = V_{W,4}$. It can be seen from the graph above that $Value\,(V_{W,4})$ is greater than the values of all its neighbours. Hence this node is a max feature and a pulse.

Steps to follow when a node is a pulse.

I. Let $V_{W,j}$ be the neighbouring node of *CNode* with the closest value to *CNode*, there will only be one neighbouring node with value closest to *CNode* since all neighbouring nodes with equal values has already been combined. In this case it can be seen that node $V_{W,3}$, with value 215, has the closest value to $V_{W,4}$.

II. Create a new node $V_{P,k}$ in the P_G with:

$$Scale = Scale\,(CNode)$$

$$Value = Value\,(CNode) - Value\,(V_{W,j})$$

III. Then for all nodes in the pulse graph, say $V_{P,p}$, connected to *CNode*, create an arc from $V_{P,p}$ to $V_{P,k}$ and delete the virtual edge from *CNode* to $V_{P,p}$. Then create a virtual edge from $V_{W,j}$ to $V_{P,k}$, and also set:

$$Scale\,(V_{W,j}) = Scale\,(V_{W,j}) + Scale\,(CNode)$$

IV. Then $V_{W,j}$ inherits all the edges and virtual edges of *CNode*, *CNode* is removed from the feature table and from the W_G. $V_{W,j}$ is added to the feature table. After this step the feature table for this example consists only of $V_{W,3}$ and the updated graphs are presented in Figure 3.7 c).

Now move on to the next node in the feature table which is of scale equal to the scale parameter n. Repeat step 4 of the algorithm until only the infinite scale, 0 node is left in the working graph. When the signal in the above example is fully decomposed we arrive at the final pulse graph presented in Figure 3.7 d). This graph is the final output of the Roadmaker's Pavage algorithm. The full Roadmaker's Pavage implementation is visually illustrated in Figure 3.8.

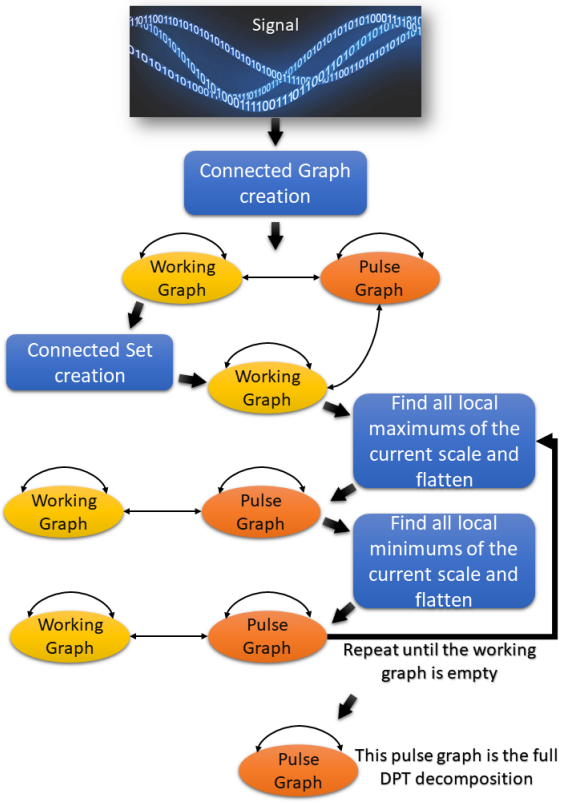


Figure 3.8: Visual illustration of the Roadmaker's Pavage implementation explained in Section 3.2.2.

3.3 Feature detection using the Discrete Pulse Transform

Feature detection is the process of segmenting a image into visually significant and disjoint segments called features. This task can be achieved using a range of techniques, with one of the most popular being convolutional neural networks [8, 24]. Here we will investigate how we can detect features in images using the DPT. The use of the DPT as a feature detection algorithms has been discussed in literature in [58, 12, 5, 52]. We will now further investigate this, as well as extend it to a graphical approach for better computational efficiency.

In [58] van der Walt provides a method for using the DPT to recognize significant segments in an image. This method uses the saliency of each pixel. Saliency is defined as the quality of being noticeable

or important, meaning a salient region is visually noticeable to the human eye [23]. The saliency of a pixel can be measured by the number of DPT pulses which contain that pixel. To define saliency for the DPT first note that a pixel is written as the sum of pulses, that is the value, $f(\mathbf{x})$, of a pixel, $\mathbf{x}$:

$$f(\mathbf{x}) = \sum_{n=1}^{N} D_n(\mathbf{x}) = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} p_{n,i}(\mathbf{x})$$

where D_n is a scale layer of scale n and $p_{n,i}$ is the i^{th} pulse in D_n. Then the saliency of a pixel $\mathbf{x}$ in an image can be defined as:

$$s(\mathbf{x}) = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} \delta_{\mathbf{x}}(p_{n,i})$$

where

$$\delta_{\mathbf{x}}(p_{n,i}) = \begin{cases} 1 & \text{if } \mathbf{x} \in p_{n,i} \\ 0 & \text{otherwise} \end{cases}.$$

Hence $\delta_{\mathbf{x}}(p_{n,i})$ indicates if pixel $\mathbf{x}$ is present in pulse $p_{n,i}$. Whilst the saliency, also called the strength of a pixel, is a strong measure of a significance, van der Walt also emphasizes that saliency on its own is not enough to select significant features. He states that another property of a feature necessary for the feature to be significant is that the pulses that contains it quickly reduces in value as we go from scale K to scale 1. Meaning we want the value of a pulse to decrease significantly as the scale of the pulse decreases. A measure that can be used for a feature's 'sharpness' [58] is

$$\eta(\mathbf{x}) = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} \frac{|p_{n,i}(\mathbf{x})|}{\sqrt{n}}.$$

Van der Walt proposes using a general significance measure of a pixel defined as

$$S(\mathbf{x}) = s(\mathbf{x}) \times \eta(\mathbf{x}) \tag{3.1}$$

so that the higher this measure the more significant the pixel.

3.3.1 Graph based feature detection using the Roadmaker's Pavage

We now show how van der Walt's S measure in Equation 3.1 for feature detection can be determined using the graph-based DPT algorithm, the Roadmaker's Pavage.

Let us consider the pulse graph obtained by the Roadmaker's Pavage algorithm in Section 3.2.2. This graph can be seen in Figure 3.7 d). In this graph we can see that each pixel is represented as a 'path' through the graph. These 'paths' make it easy to calculate the significance measure S from Equation 3.1. Let each pixel store the value and scale of each node it encounters on its 'path'. If we follow the 'path' of pixel number 3 in the graph, we see that it encounters 3 nodes on its way. The first node it encounters is $V_{P,5}$ which has value, scale pair $(33, 2)$, it then encounters $V_{P,6}$ with pair $(148, 3)$ and last $V_{P,7}$ with pair

$(67, 4)$. The saliency of a pixel is the number of nodes it encounters on its path, so pixel 3 has a saliency of 3. Next to calculate the 'sharpness' of a pixel we calculate

$$\eta(\mathbf{x}) = \sum_{V \in \text{path}} \frac{\text{Value}(V)}{\sqrt{\text{Scale}(V)}}.$$

Hence for pixel 3 we get

$$\begin{aligned}\eta(\mathbf{x}) &= \frac{33}{\sqrt{2}} + \frac{148}{\sqrt{3}} + \frac{67}{\sqrt{4}} \\ &= 142.28236,\end{aligned}$$

so that the S measure for this pixel is $S = 142.28236 \times 3 = 426.84709$. Following the same steps for the other pixels in this graph we obtain the S significance map in Figure 3.9.

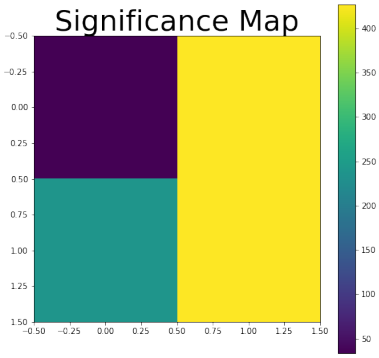


Figure 3.9: Significance map of signal decomposed in Figure 3.2. This significance map is based on the S measure of van der Walt [58].

The next step in this feature detection method is to select the most significant features. This will be done using a parameter α which indicates which percentage of the most significant features we want to extract. In Figure 3.10 we illustrate the features extracted in an image of a wooden shingles roof from USC-SIPI texture image database [1]. We see that some of the features extracted highlights the edges of some of the shingles and some of the other features highlight details such as stains and scratches on the shingles. The steps of this algorithm is summarized below:

I Using the Roadmaker's Pavage algorithm obtain the graphical DPT decomposition of the image.

II Using the graph based method, calculate the S measures of each pixel.

III Store the locations of the αN most significant pixels, where N is the total number of pixels in an image.

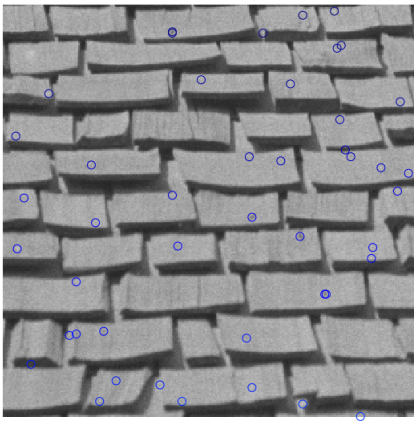


Figure 3.10: Significance features extracted by the graphical Discrete Pulse Transform feature extraction method.

3.4 Texture extraction using the Discrete Pulse Transform

In this section we illustrate the ability of Discrete Pulse Transform to extract textures from an image. As discussed previously in this chapter the DPT decomposes an image into scale layers $\{D_1, D_2, ..., D_N\}$, where D_n is a scale layer of scale n. We have also seen that a image, f, can be reconstructed from this decomposition by taking the sum of the scale layers, that is

$$f = \sum_{n=1}^{N} D_n = \sum_{n=1}^{N} \sum_{i=1}^{\gamma(n)} p_{ni},$$

where p_{ni} is the i^{th} pulse of scale n. We can also obtain a partial reconstruction of the image using scale layers of scales $\mathcal{S} = \{n_1, n_2, ..., n_s\}$. The partial reconstruction of image f is

$$f_{\mathcal{S}} = \sum_{n \in \mathcal{S}} D_n.$$

Fabris-Rotelli and Stein [11] discuss the use of partial reconstructions to highlight the textures in images. They use the frequencies of pulses at different scales to illustrate that majority of the details in an image is captured in pulses of scales 1 to 10. In Figure 3.11 we show that majority of the information in the image considered is represented by pulses of small scale. This is shown for other images in [10, 11].

Fabris-Rotelli [10] proposed extracting a pulse map which represents the texture information in an image. This is done by first calculating the fraction of information in an image represented by pulses of scale n or less, for all scales n. Then the smallest scale n_β such that pulses of scale n_β or less represent at least a fraction β of the information in an image is found. The pulse map is the partial reconstruction given as

$$f_\beta = \sum_{n=1}^{n_\beta} D_n.$$

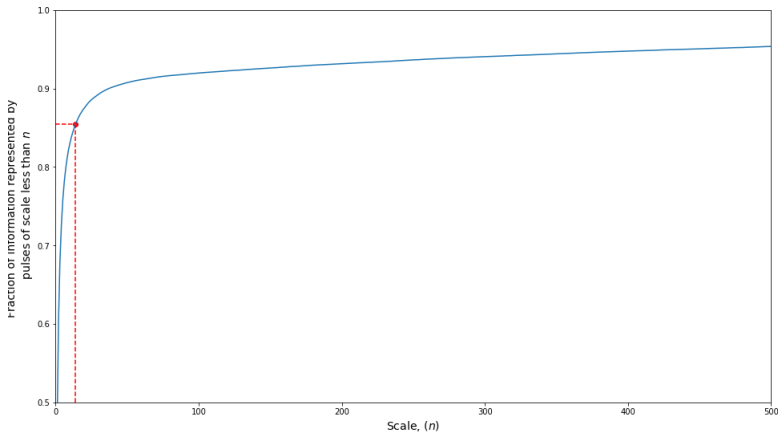


Figure 3.11: Illustration of the fraction of information in an image represented by pulses of scale n or less. The image used for this illustration is the image from Figure 3.10. The red dotted lines indicate that 85% of the information is represented by scales 14 or less.

We aim to extract the texture details as the spatial information to model since it has been shown that using textures to match features works well. Methods such as convolutional neural networks and filter banks use texture filters to describe features. The use of the texture filters in Figure 3.12 for feature detection is described by Olshausen and Field [37]. Olshausen and Field also highlight the importance of using texture based techniques in natural image processing.

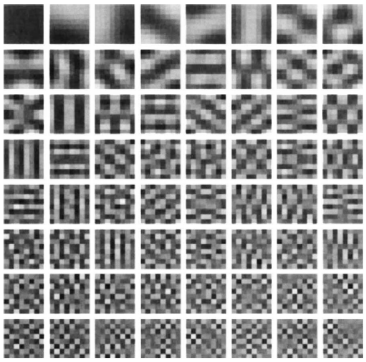


Figure 3.12: Filter bank of textures from Olshausen and Field 1996 illustrating the importance of texture filters [37].

The steps of our proposed algorithm are summarized below:

I Using the Roadmaker's Pavage algorithm obtain the graphical DPT decomposition of the image.

II From the decomposition calculate the number of pulses at each scale i, namely $\gamma(i)$. Using these

counts calculate the cumulative number of pulses of scale n or less,

$$\kappa(n) = \sum_{i=1}^{n} \gamma(i).$$

III Calculate the fraction of information represented by pulses of scale n or less, $\frac{\kappa(n)}{\kappa(N)}$. Where N is the number of pixels in the image.

IV Find the smallest scale n_β such that pulses of scale n_β or less represent at least a fraction β of the information in an image. That is

$$\frac{\kappa(n_\beta)}{\kappa(N)} \geq \beta.$$

V Return the pulse map

$$f_\beta = \sum_{n=1}^{n_\beta} D_n.$$

An example of a pulse map extracted by this method is shown in Figure 3.13. In this figure we see that the edges of the shingles as well as the textures on each shingle from the wood shingle roof image is extracted by this method. We also see that the extracted textures are similar to the texture banks used by Olshausen and Field in their texture filters shown in Figure 3.12. In Table A.2 in the appendix the other examples of the textures extracted by this method are shown. From this table we can see that this method is efficient at extracting the texture information from an image.

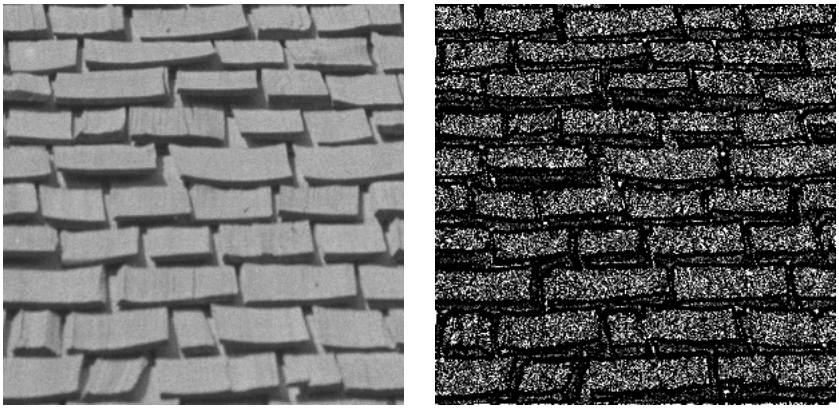


Image Pulse map

Figure 3.13: Pulse map extracted using the Discrete Pulse Transform texture extraction method.

3.5 Concluding remarks

In this chapter the details of scale-space methods and in particular the Discrete Pulse Transform were discussed. We have also discussed a graphical DPT implementation, the Roadmaker's Pavage, in detail,

and have shown how the Roadmaker's Pavage can be used as a graphical multiscale feature detection method. We then also illustrated how the DPT decomposition can be used to extract textures of different scales from an image. We will next investigate the combination of this multiscale graphical feature detection method, the multiscale texture extraction method and the Direct Sampling Feature Matching method.

Chapter 4

Feature detection and matching with applications

In Chapter 2 we proposed a new method for feature matching using Direct Sampling. This method does not make use of any traditional feature detection tools. Then in Chapter 3 we discussed a multiscale graphical feature detection tool. This method is based on the Roadmaker's Pavage implementation of the DPT and van der Walt's significance measure. We also discuss a texture extraction method which uses a partial DPT reconstruction to emphasise textures. In this chapter we aim to combine these two algorithms into a novel multiscale spatial feature matching algorithm. We test the strength of this method using the USC-SIPI texture database [1]. We also investigate the use of this algorithm as a feature localisation method for object tracking. The code for this work is available at a GitHub repository (available at: `https://github.com/CarelvN/Multiscale-spatial-modeling-with-applications-in-image-analysis`).

4.1 Texture distance comparison using Direct Sampling Feature Matching

In Section 2.3 we proposed a novel feature matching and localisation algorithm which uses the links created by Bunch-pasting Direct Sampling, called Direct Sampling Feature Matching (DSFM). The algorithm can also determine the distance between two images. Here we investigate the performance of this distance when comparing images in the USC-SIPI texture image database [1]. This USC-SIPI texture database consists of labelled texture images. Some examples of these images are provided in Figure 4.1. The full database can be found in Table A.1 in the appendix. For the comparison we will be using the classes within this dataset. As seen in Figure 4.1 there are 21 different classes in the dataset, the full dataset is shown in Table A.1 in the appendix. The comparison is done as follows. A randomly chosen image from one of the classes in the database is chosen to be matched using the DSFM method. Only the 14

Figure 4.1: A sample of the USC-SIPI texture image database.

classes with at least 3 unique examples in the database will be considered for this performance test. Then three randomly chosen images from the same class are selected from the database. These images are then iteratively put into the DSFM as the image to be matched to.

Next another 3 images are chosen randomly, this time from any class other than the one under consideration. These images can be picked from any of the 21 classes in the database. These images are also iteratively put into the DSFM as the image to be matched to.

The DSFM method then determines the distance between each pair of images. This distance is the distance related to the weakest link created by the DSFM method. Hence the distance between two images is

$$\max_{l \in \mathcal{M}} \mathcal{D}(l),$$

where $\mathcal{M}$ is the set of links created between two images by DSFM and $\mathcal{D}(l)$ is the distance between the neighbourhoods of the pixels linked by link l. Meaning after comparing the originally chosen image to 6 images we have 6 distances, 3 within class distances and 3 alternate class distances. We store the average of the 3 within class distances and the average of the 3 alternate class distances.

For good performance we want the difference, within class average distance minus alternate class average distance, to be significantly larger than 0. Showing us that the algorithm can distinguish between

textures. This method is described visually by the diagram in Figure 4.2 where we can see that we compare a labelled images to images within the same class, as well as to images in an alternate class to obtain within and alternate class distances.

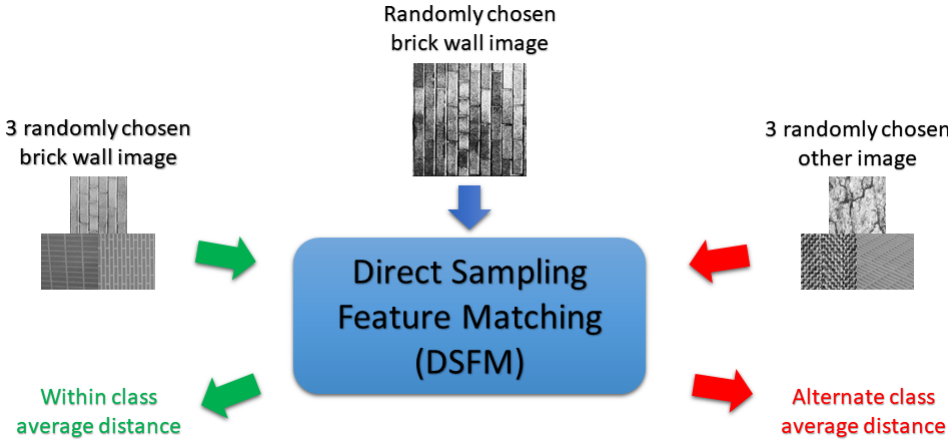


Figure 4.2: Distance comparison using the Direct Sampling Feature Matching method.

To comprehensively test the DSFM method we perform the distance test illustrated in Figure 4.2 using a comprehensive set of parameters. We will be testing thresholds $t = 0.2, 0.25, 0.3, 0.35, 0.4$ for each $f = 0.8, 0.85, 0.9$, and we will test these parameters for each $k = 5, 6, 7, 8$. Where k is the width of a square neighbourhood window used for DSFM in this application. That is we will be performing 60 distance tests for each of the labels shown in Figure 4.1. For this test we will only be using the classes with at least 3 unique examples of that class in the dataset. The overall accuracy will be measured by the fraction of labels for which the within class average distance is less than the alternate class average distance. This is calculated as follows. For the within class average distance of class i, $\mathcal{D}_W(i)$, and the alternate class average distance of class i, $\mathcal{D}_A(i)$, the accuracy over C classes is

$$\text{Accuracy} = \frac{1}{C}\sum_{c=1}^{C} I(\mathcal{D}_W(c), \mathcal{D}_A(c)), \tag{4.1}$$

where I is the indicator function

$$I(\mathcal{D}_W(c), \mathcal{D}_A(c)) = \begin{cases} 1 & \text{if } \mathcal{D}_W(c) < \mathcal{D}_A(c) \\ 0 & \text{otherwise} \end{cases}.$$

In Figures 4.3 - 4.6 we plot the within class and alternate class average distances for each class. This is done for a range of different parameter values. These plots also use a standardised distance, which is the distances divided by the maximum distance, to make tests using different distance thresholds comparable. This is necessary because the t and f parameters affect the magnitude of these distances. We also indicate the difference between these average distances using a green line if the within class distance is smaller than the alternate class distance, and a red line if the opposite is true. These figures provide us with an in-depth

view of the effects of different parameter combinations on specific textures. In Figure 4.7 we also plot the overall accuracies measured by Equation 4.1 of the DSFM algorithm using different parameter values. These plots can be used to investigate the overall performance of the DSFM algorithm. It is however important to consider the in-depth performance analysis in combination with the overall performance to fully understand the effects of parameters and the abilities of this method.

We see from these results that the DSFM algorithm performs relatively well. We now go a step further to increase performance of the image matching method proposed. Up to now all the performance tests randomly pick nodes to mark as empty. We now investigate the improvement to this method by using the significant features from the DPT in Chapter 3. We proposed this method to add in a feature detection component to this method. We further proposed focussing on the textures in images, by using the textures extracted by the DPT as the spatial structure to model. Since it has been shown that using textures improves performance when working with natural images [11, 37].

4.2 Multiscale spatial modelling of features using the Discrete Pulse Transform and Direct Sampling Feature Matching

In Section 2.3 we proposed a new application of the Bunch-pasting Direct Sampling algorithm called Direct Sampling Feature Matching (DSFM). We can achieve even better results by adding in the DPT feature detection and texture extraction methods. This is done by detecting features using the graphical Roadmaker's Pavage DPT. We will also utilise the textures extracted using the DPT as the spatial structures to be matched. These pulse maps (images containing textures extracted) will be used as the images (spatial structures) for the DSFM method, the empty nodes for the 'simulation grid' will be selected at the locations of the significant features detected. We call this novel method Multiscale Spatial Feature Matching (MSFM).

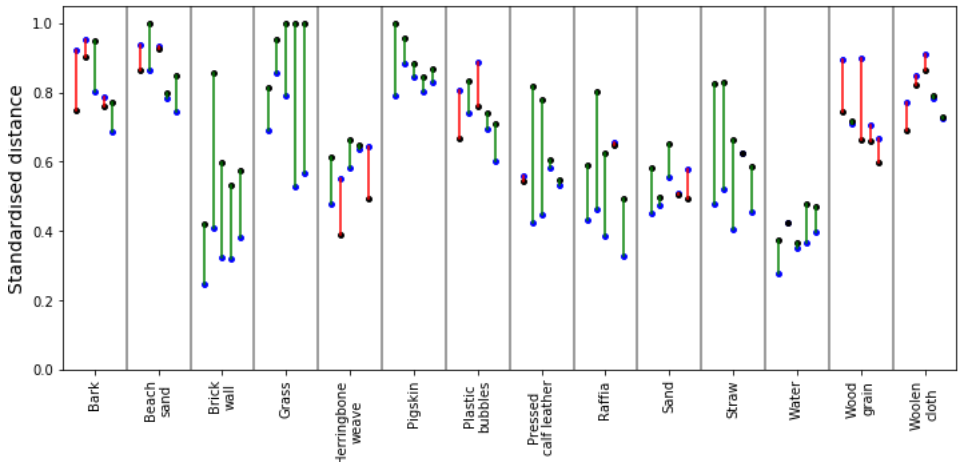
The input of this algorithm as for the DSFM is two images. The algorithm then has 5 tuning parameters. The first tuning parameter is α, this parameter indicates which percentage of the most significant features to use for matching. The β parameter indicates the fraction of information we want to extract from the image, and t, f are the parameters of the Bunch-pasting Direct Sampling algorithm. The neighbourhood window used for the Bunch-pasting Direct Sampling algorithm in this application will be a square window of width k. The steps of this algorithm is summarized below:

I Using the Roadmaker's Pavage algorithm obtain the graphical DPT decomposition of the two images.

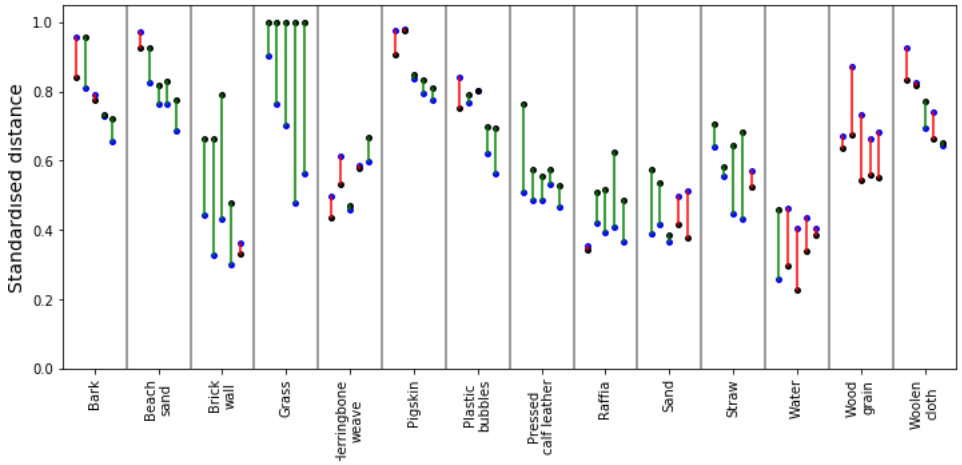
II Using the graph based method, calculate the S measure of each pixel in the image to be matched.

III Store the locations of the αN most significant pixels, where N is the total number of pixels in this image.

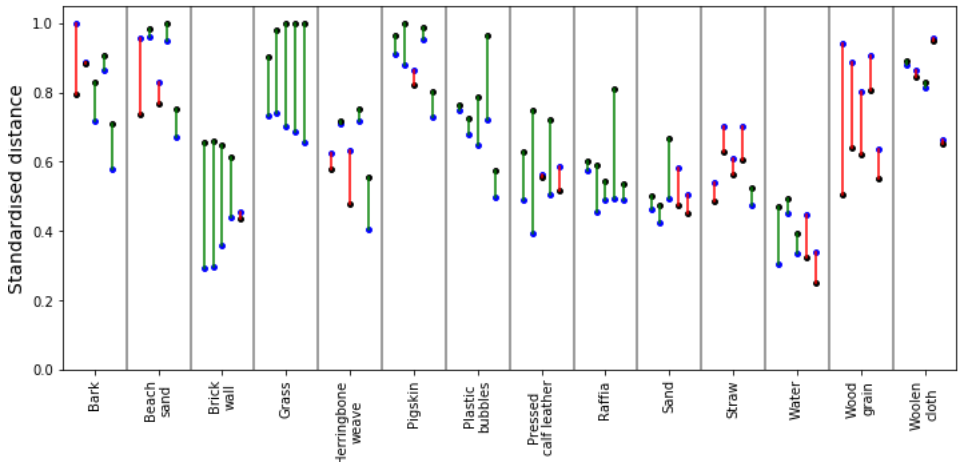
IV For each of the two images:



$f = 0.8$

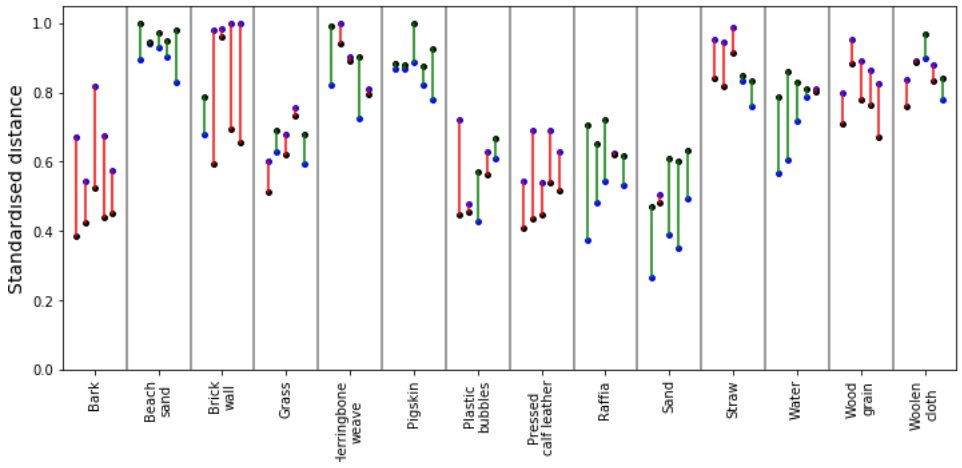


$f = 0.85$

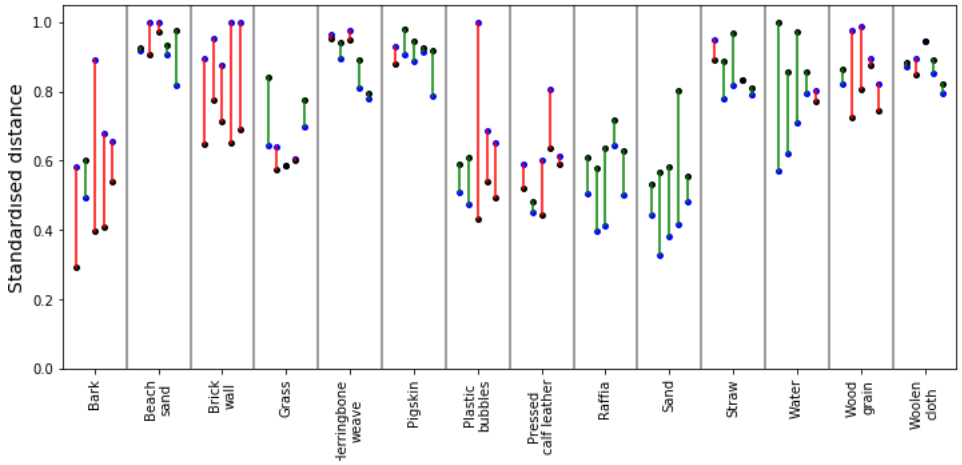


$f = 0.9$

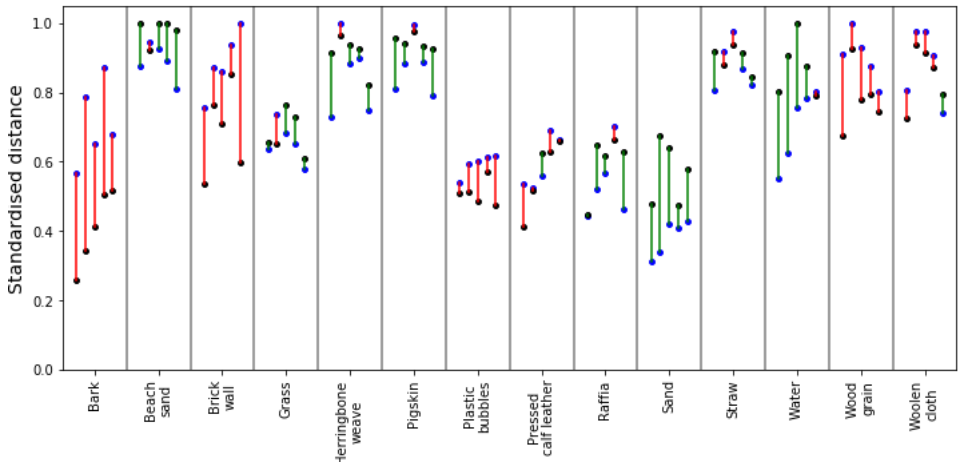
Figure 4.3: Within and alternate label distance comparisons for the DSFM method using neighbourhood size $k = 5$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$

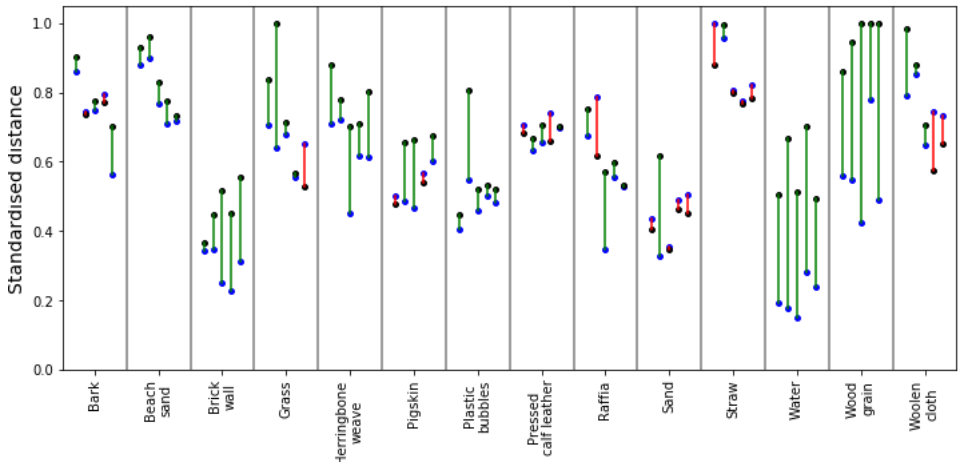


$f = 0.85$

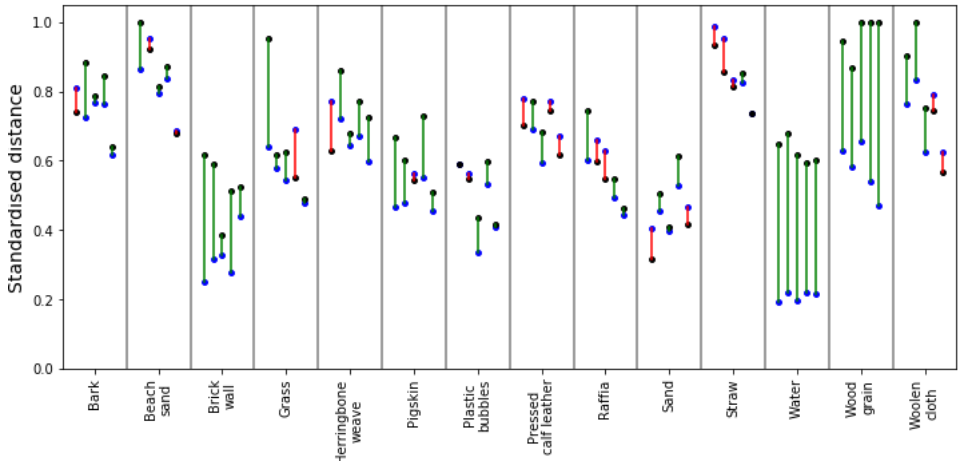


$f = 0.9$

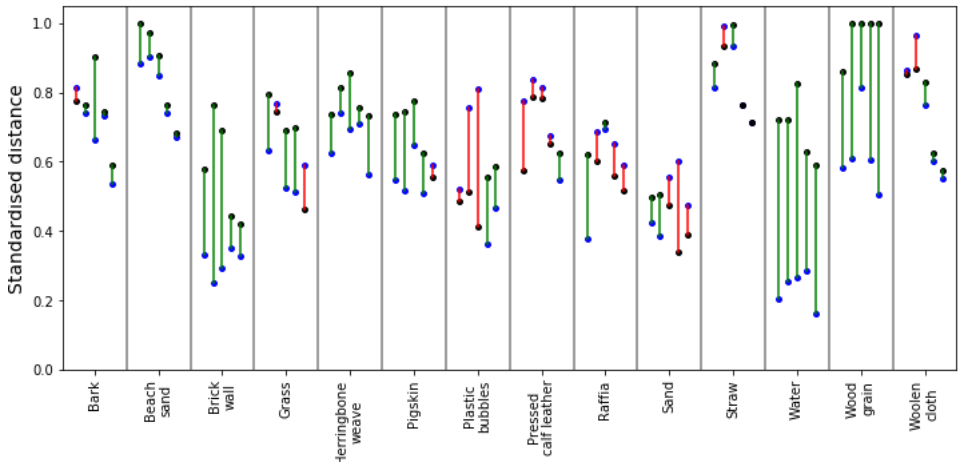
Figure 4.4: Within and alternate label distance comparisons for the DSFM method using neighbourhood size $k = 6$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$

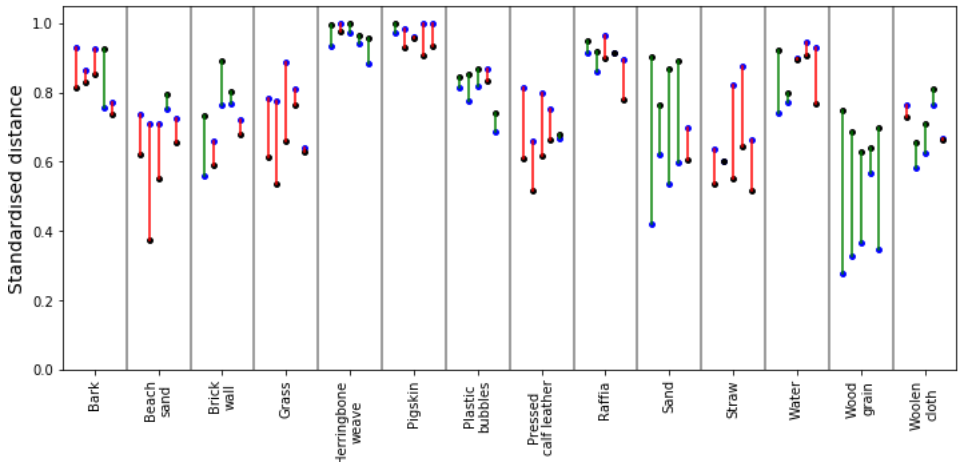


$f = 0.85$

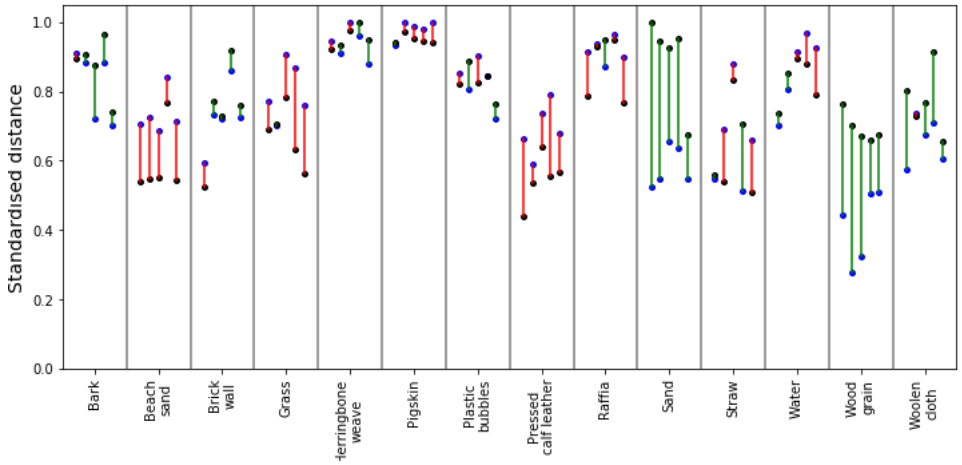


$f = 0.9$

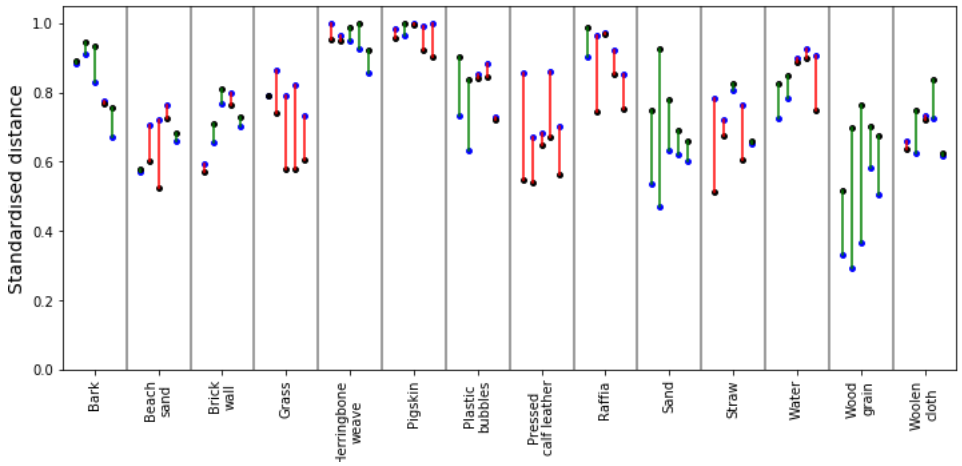
Figure 4.5: Within and alternate label distance comparisons for the DSFM method using neighbourhood size $k = 7$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$



$f = 0.85$



$f = 0.9$

Figure 4.6: Within and alternate label distance comparisons for the DSFM method using neighbourhood size $k = 8$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.

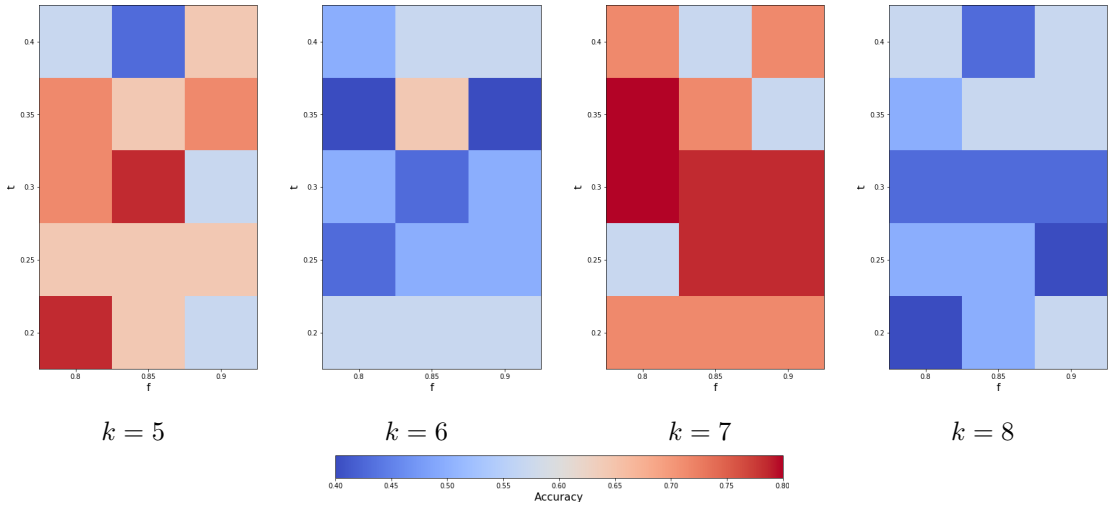


Figure 4.7: Accuracy of the DSFM algorithm.

i From the DPT decomposition calculate the number of pulses of scale i, $\gamma(i)$. Using these counts calculate the cumulative number of pulses of scale n or less,

$$\kappa(n) = \sum_{i=1}^{n} \gamma(i).$$

ii Calculate the fraction of information represented by pulses of scale n or less, $\frac{\kappa(n)}{\kappa(N)}$. Where N is the number of pixels in the image.

iii Find the smallest scale n_β such that pulses of scale n_β or less represent atleast a fraction β of the information in an image. That is

$$\frac{\kappa(n_\beta)}{\kappa(N)} \geq \beta.$$

iv Return the pulse map

$$f_\beta = \sum_{n=1}^{n_\beta} D_n.$$

V Set the pulse map of the image to match to as the 'training image'.

VI Set the significant pixels identified in III as empty pixels in the pulse map of the image to be matched.

VII Set this pulse map with its empty nodes as the 'simulation grid'.

VIII Perform Bunch-pasting Direct Sampling (BDS).

IX Determine the distance between the two images. This distance is the distance related to the weakest link created. Hence the distance between two images is

$$\max_{l \in \mathcal{M}} \mathcal{D}(l),$$

where $\mathcal{M}$ is the set of links created between the two images and $\mathcal{D}(l)$ is the distance between the neighbourhoods of the pixels linked by link l.

4.2.1 Texture classification application

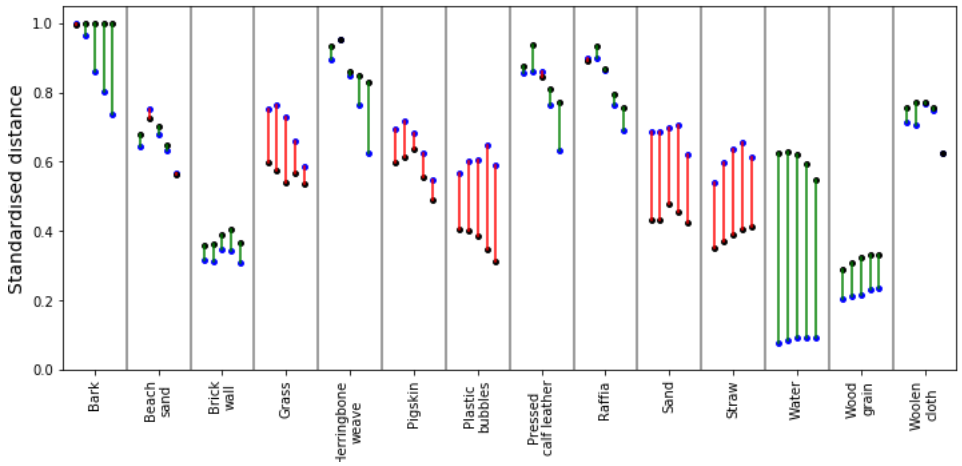
We now test the performance of this algorithm using the same distance comparison method as used in Section 4.1. In this case we use the MSFM algorithm to calculate distances rather than the DSFM algorithm. In Figures 4.8 - 4.11 we plot the within class and alternate class average distances for each class. This is done for a range of different parameter values as in Section 4.1. We also indicate the difference between these average distances using a green line if the within class distance is smaller than the alternate class distance, and a red line if the opposite is true. These figures provide us with an in-depth view of the effects of different parameter combinations on specific textures. In Figure 4.12 we plot the overall accuracies according to Equation 4.1 of the MSFM algorithm using different parameter values. These plots can be used to investigate the overall performance of the DSFM algorithm. It is however important to consider the in-depth performance analysis in combination with the overall performance to fully understand the effects of parameters and the abilities of this method. We see from these results that the MSFM algorithm performs relatively well. We will next discuss the performance of these two matching algorithms.

4.3 Discussion

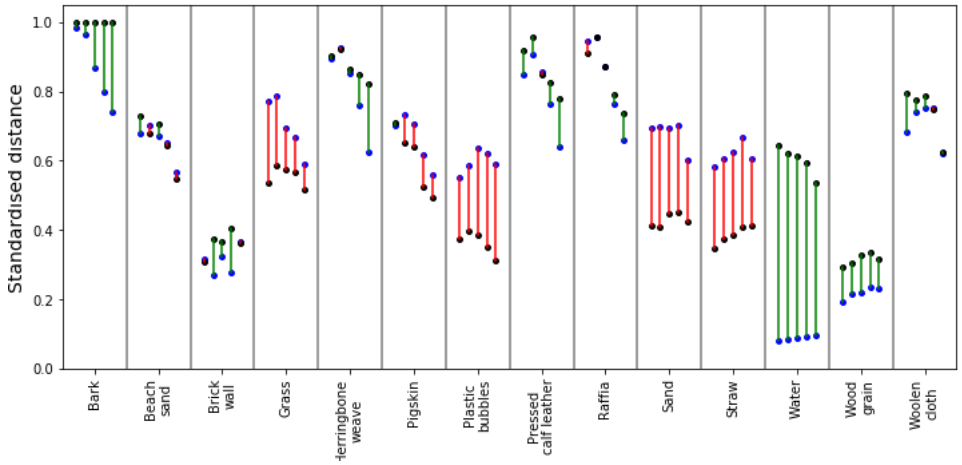
We begin this discussion with an in-depth analysis of the performance of the DSFM and MSFM algorithms. We look at the performance on different types of textures and the effects of different parameter pairings. For this in-depth analysis we will be considering the difference between the within class distances and alternate class distances, illustrated in Figures 4.3 - 4.6 and Figures 4.8 - 4.11.

The first texture we consider which highlights some interesting performance is the brick wall texture. We observe in Table A.1 in the appendix that there are numerous examples of this texture, where we observe brick walls at different scales and rotations. We also observe some textures like metal grates and wood shingle roofs which are visually similar to brick walls. For the DSFM algorithm we observe slightly erratic performance on the brick wall textures. For some neighbourhood window sizes we observe insignificant differences in distances and even alternate class distances which are significantly less than the within class distances. We do however observe that at neighbourhood window widths $k = 5$ and $k = 7$ the difference in distances is significant and positive. This may be an indication that the neighbourhood size affects the ability to distinguish between brick walls and other textures. This may be due to the different sizes of bricks. When we look at the MSFM results in Figures 4.8 - 4.11 we see a clearer trend regarding this texture. We see that as the neighbourhood size increases the differences between the distances become more significant leading to the algorithm being more accurate at detecting this texture.

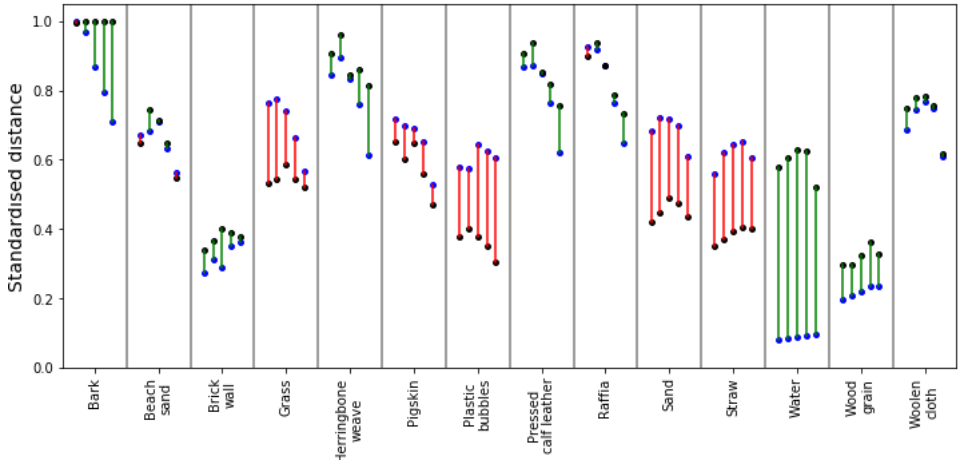
Bark is another texture which has performed relatively erratically without much significance or accuracy during the DSFM testing. We do suspect that this may be due to the unique patterns on the bark as well as the similarity of bark to other natural patterns such as wood grain. We see a signifi-



$f = 0.8$

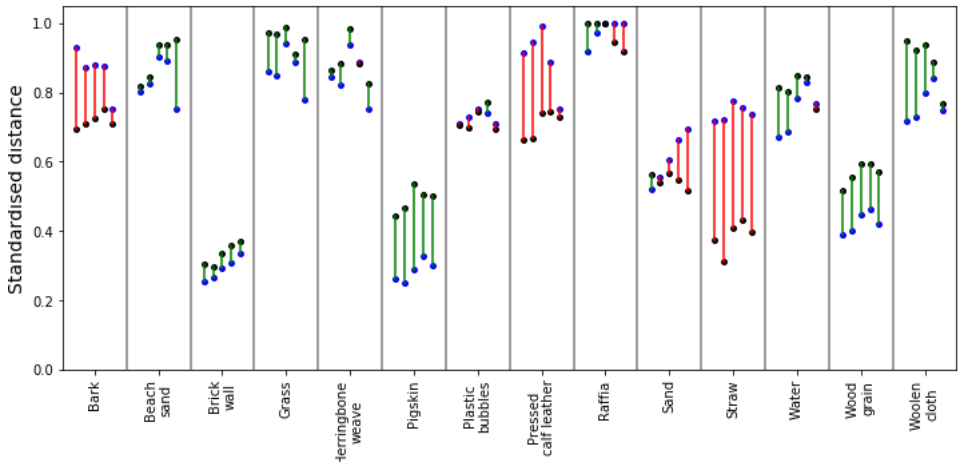


$f = 0.85$

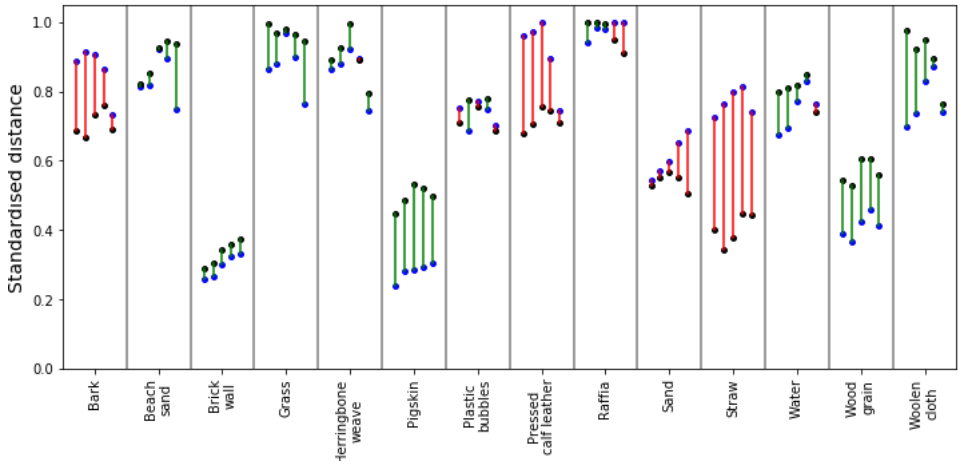


$f = 0.9$

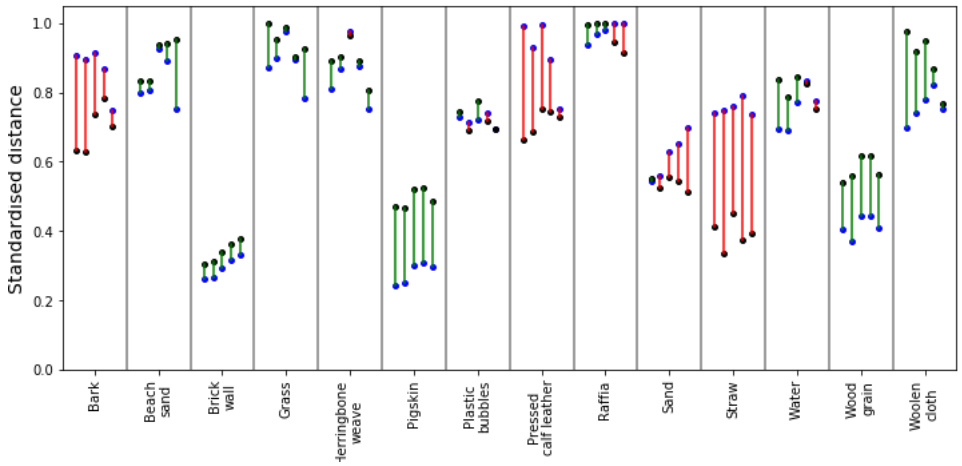
Figure 4.8: Within and alternate label distance comparisons for the MSFM method using neighbourhood size $k = 5$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$

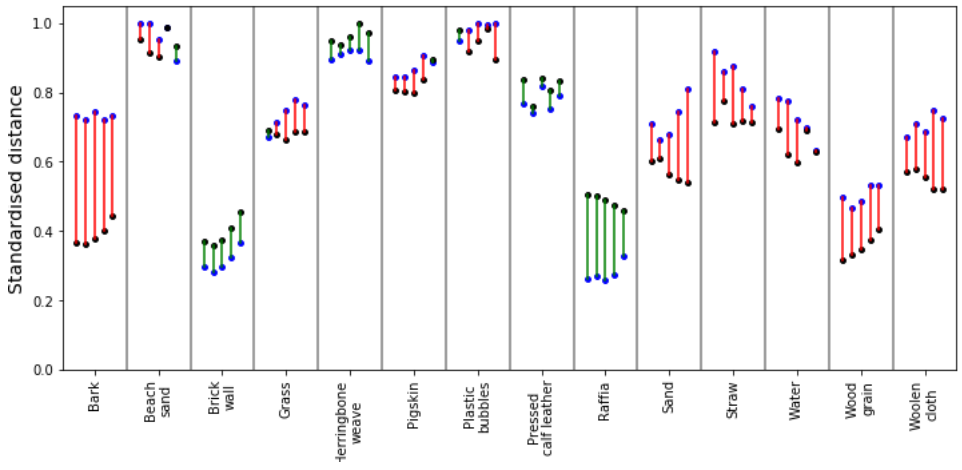


$f = 0.85$

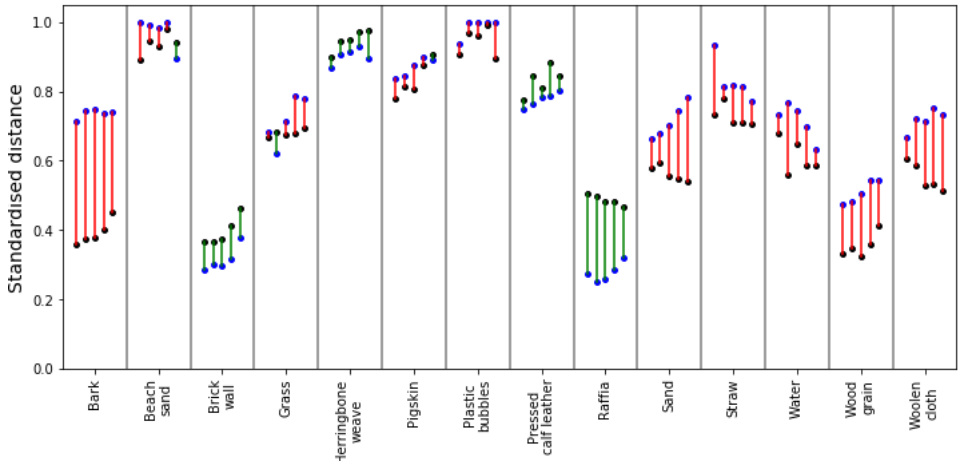


$f = 0.9$

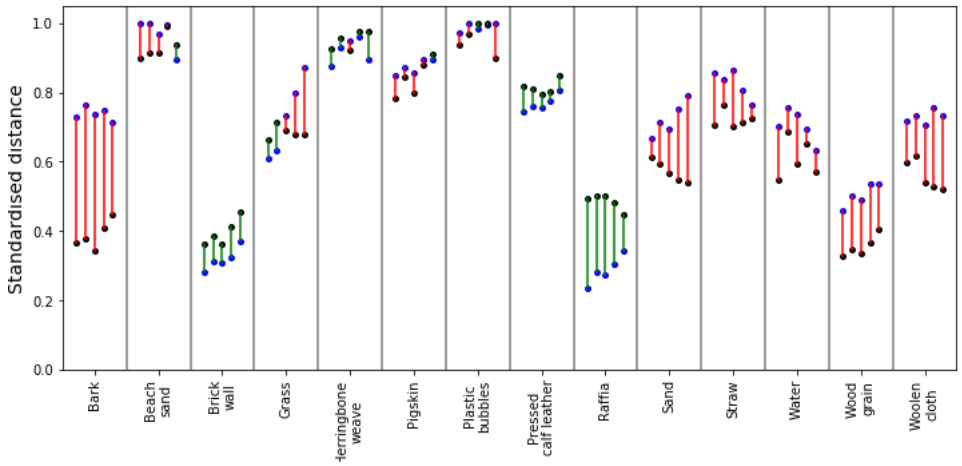
Figure 4.9: Within and alternate label distance comparisons for the MSFM method using neighbourhood size $k = 6$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$

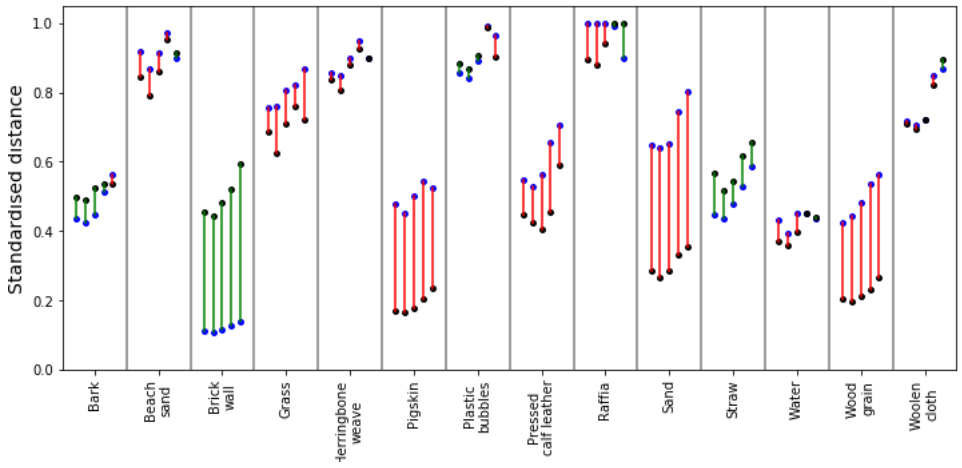


$f = 0.85$

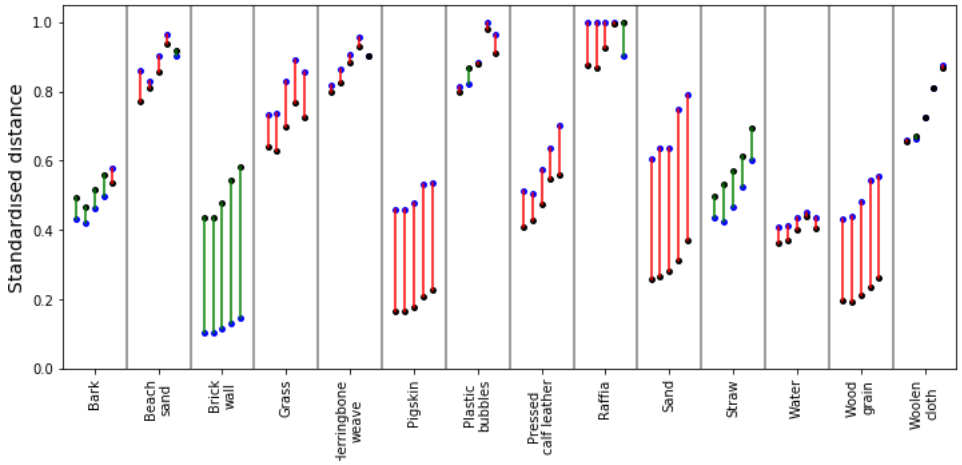


$f = 0.9$

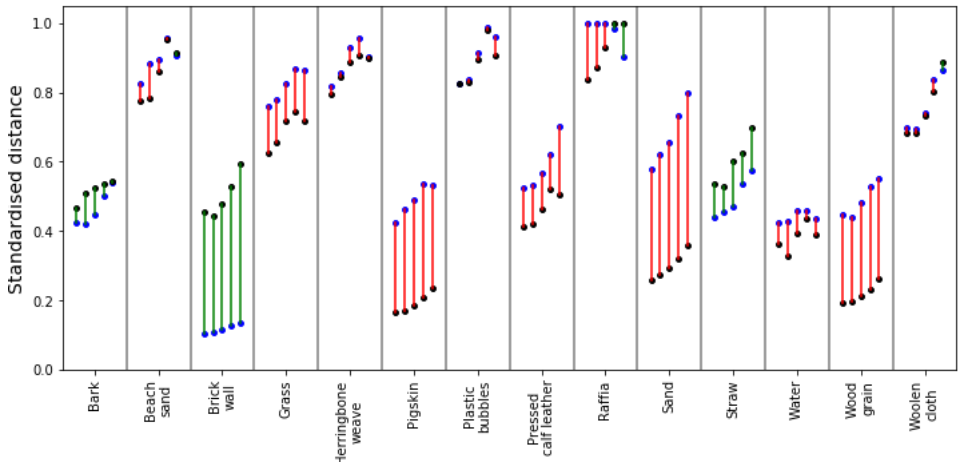
Figure 4.10: Within and alternate label distance comparisons for the MSFM method using neighbourhood size $k = 7$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.



$f = 0.8$



$f = 0.85$



$f = 0.9$

Figure 4.11: Within and alternate label distance comparisons for the MSFM method using neighbourhood size $k = 8$. For each class the average distances for threshold values 0.4, 0.35, 0.3, 0.25, 0.2 are plotted respectively from left to right within each label.

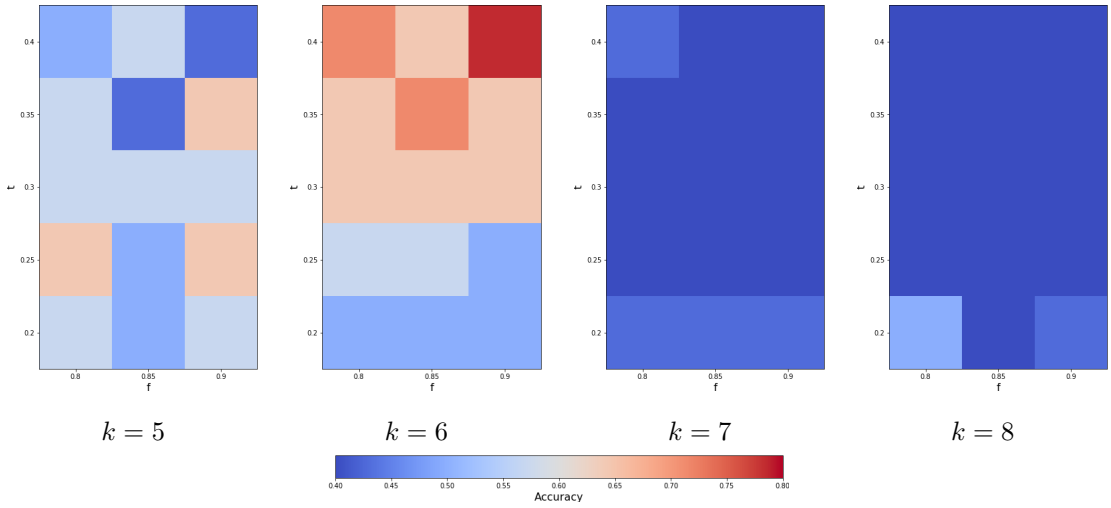


Figure 4.12: Accuracy of the MSFM algorithm.

cant improvement in accuracy when the MSFM method was applied. This is particularly at the smaller neighbourhood windows when using smaller (more strict) threshold parameters closer to $t = 0.2$. This performance deteriorated when using larger neighbourhood windows. We suspect that this performance is due to the fact that the finer details and textures extracted by the DPT can be matched much more accurately than the original image information. The finer details extracted by these pulses are however lost when matching using a larger neighbourhood window.

Water and wood grain are two textures which was also found to be reliant on the neighbourhood window size when matching. The overall performance for these textures does not yield significant or positive results most of the time. At a neighbourhood window width of $k = 7$ the DSFM algorithm does achieve some positive accurate results for these textures. The MSFM algorithm using small neighbourhood windows achieves more accurate results consistent for these two textures. This once again highlights the fact that the textures extracted by the DPT matched at a smaller neighbourhood size can perform well consistently with some difficultly discriminating between textures. On the contrary there are also textures such as straw, which also indicated insignificant and negative distance differences when using the DSFM method indicated that the MSFM can be much stronger when matching textures. The straw texture, very difficult to discriminate from textures such as grass, has consistent positive distance differences using MSFM with a neighbourhood window of width $k = 8$.

We also witnessed that MSFM was able to obtain significant positive distance difference distances when considering a texture which proved to be extremely difficult to discriminate from other textures, such as a woolen cloth, due to how similar this texture is visually to other fabric or natural textures such as herringbone weave or other skin. Some similar textures such as pressed calf leather and raffia also proved to be very difficult to discriminate from other textures and for these textures the MSFM method also managed to find some positive significant distance differences. For all these textures the DSFM algorithm

could not consistently find significant positive distance differences. There were however textures such as beach sand, herringbone weave, plastic bubbles, grass and pigskin which both the DSFM and the MSFM methods could not perform well on. This is mainly because these textures are too similar visually to other textures and hence it was not possible to discriminate between them. This is most likely because the 'beach sand' texture was too similar to 'sand' or other rough textures like 'grass'. The 'herringbone weave' in turn is too similar to textures like 'raffia' or 'woolen cloth'. These different textures are in some cases even difficult for the human eye to discriminate quickly between.

These results illustrate that the addition of the multiscale feature detection and texture extraction tools has given the matching algorithm the ability to discriminate between some very difficult textures. This extra accuracy was added by using only texture information rather than a full image, and by matching significant features points rather than randomly selected points. We cannot expect perfect performance as this is most likely even impossible for the human eye.

We now also shortly discuss the overall performance of these algorithms. We do this using the plot of the accuracy measures measured by Equation 4.1 for all the considered parameter combinations. We see in Figure 4.7 that the DSFM algorithm performs best using a neighbourhood window of width $k = 7$. We do however see that the performance does not have any visible relationship to the neighbourhood size when looking at the overall performance. From these plots we also see that the threshold and fraction of windows to scan, the t, f parameters, do not significantly affect performance however it does seem like lower f such as $f = 0.8$ and $f = 0.85$ do perform better than the stricter $f = 0.9$. When considering the overall performance of the MSFM algorithm in Figure 4.12 we immediately notice that using neighbourhood windows which are too large results in loss of the texture details extracted by the DSFM. This also highlights that the parameters of the method needs to be fine tuned based on the textures one needs to discriminate between as there are some textures which perform better with a larger neighbourhood window. At $k = 6$ MSFM performs better for larger thresholds. This is mainly due to the fact that the texture images are 0 at all pixels other than the textures extracted, making it hard to find very small distances between windows. From the accuracy plots we can also see that there is a definite indication to which parameter settings achieves better global accuracy for the MSFM method, the accuracy results of the DSFM method is harder to interpret. This means that it would be easier to fit parameters for the MSFM method compared to the DSFM method.

From the discussion above we see that the MSFM method proposed performs relatively well for the matching of texture images. It does however still have weaknesses when comparing similar textures. The method is also location invariant due to the scanning windows used, this is a strength this method shares with Convolutional neural networks. The method is however unfortunately not completely scale and shape invariant. The DPT feature detection and texture extraction method is a scale and shape invariant method due to the adaptive size and shape of the neighbourhood filters. Unfortunately the Direct Sampling method is not shape or scale invariant, this is due to the fixed shape and size of the

neighbourhood filters used.

We also see that the method needs unique parameters for varying textures, this makes it difficult to choose a reliable and consistent set of parameters of an application. From the discussion it was also observed that the performance of this method is reliant on the parameter choice. With a large amount of parameters to choose from such as neighbourhood shapes and sizes, significance thresholds, etc. it can become rather tedious to apply this method to problems. It will be necessary to perform a large grid search to find optimal parameters. Although this introduces more complexity it makes the method more difficult to apply. Another prevalent weakness of this method is that it is computationally intensive. Unlike pre-trained models such as Convolutional neural networks this model needs to perform very computationally expensive operations during the prediction or application phase. This fact makes it impractical to apply the model in real time.

Chapter 5

Conclusion

We have seen in this work that a multiscale spatial statistics method can perform well at discriminating between images of different textures. The power of the DPT as a texture extractor and feature detection method has also been highlighted. We have discussed the details of spatial simulation with application to image simulation. We have also shown that spatial image simulation can be used for feature matching. We discussed the details of the Discrete Pulse Transform multiscale tool and its abilities to identify features using the pulse decomposition. We have also introduced a new fast and efficient graphical feature detection method, making use of the Roadmaker's Pavage, which finds the significant 'paths' to identify significant pixels. We showed how the proposed novel multiscale spatial modeling method can discriminate between some challenging textures.

In future work this algorithm could possibly be improved in the following ways. One can also investigate a selected scale-space DPT decomposition where the information in the image is used to automatically decide which scales to select. The use of adaptive scales based on the information in the area of the image in a similar fashion to the adaptive median filter [7] can also be tested. One can also investigate the use of adaptive shape and size neighbourhood filters for Bunch-pasting Direct Sampling, as is the approach of the DPT. The use of other distance metrics such as the Structural Similarity Index (SSIM) [59] to measure the distance between neighbourhood windows for the Bunch-pasting Direct Sampling can also be tested. This could improve on using the Euclidean norm because it will consider the spatial structures together with the values of the pixels rather than just the pixel values alone. Other matching methods, besides Direct Sampling, for the textures and features extracted by the DPT can also be considered. This method could in future be tested as a texture classification algorithm.

The contributions made are:

1. The development of a novel multiscale spatial modeling approach to feature matching.

2. The developed a graphical feature detection method which uses the paths through a graph to locate significant pixels.

3. We have shown the importance of considering textures when attempting to discriminate between different images.

Bibliography

[1] The USC-SIPI Image Database, 1977. [Online] http://sipi.usc.edu/database/.

[2] EH Adelson, CH Anderson, JR Bergen, PJ Burt, and JM Ogden. Pyramid methods in image processing. *RCA Engineering Laboratories Technical Journal*, 29(6):33–41, 1984.

[3] F Alhwarin, C Wang, D Ristic-Durrant, and A Gräser. Improved SIFT-Features Matching for Object Recognition. In *BCS International Academic Conference*, pages 178–190, 2008.

[4] S Alpert, M Galun, A Brandt, and R Basri. Image segmentation by probabilistic bottom-up aggregation and cue integration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):315–327, 2012.

[5] R Anguelov and IN Fabris-Rotelli. LULU operators and Discrete Pulse Transform for multidimensional arrays. *IEEE Transactions on Image Processing*, 19(11):3012–3023, 2010.

[6] H Bay, A Ess, T Tuytelaars, and L Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[7] R Bernstein. Adaptive nonlinear filters for simultaneous removal of different kinds of noise in images. *IEEE Transactions on Circuits and Systems*, 34(11):1275–1291, 1987.

[8] F Cao, Y Liu, and D Wang. Efficient saliency detection using convolutional neural networks with feature selection. *Information Sciences*, 456:34–49, 2018.

[9] XL Dai and J Lu. An object-based approach to automated image matching. In *Geoscience and Remote Sensing Symposium, 1999. IGARSS'99 Proceedings. IEEE 1999 International*, volume 2, pages 1189–1191. IEEE, 1999.

[10] IN Fabris-Rotelli. LULU operators on multidimensional arrays and applications. Master's thesis, University of Pretoria, 2009.

[11] IN Fabris-Rotelli and A Stein. Inhomogeneous spatial modelling of DPT pulses for marine images. *Spatial Statistics*, 2018.

[12] IN Fabris-Rotelli and GG Stoltz. On the leakage problem with the Discrete Pulse Transform decomposition. In *Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*, page 179, 2012.

[13] IN Fabris-Rotelli and SJ Van der Walt. The Discrete Pulse Transform in two dimensions. *Twentieth Annual Symposium of the Pattern Recognition Association of South Africa*, 2009.

[14] J Fourier. *Theorie Analytique de la Chaleur*. Chez Firmin Didot, 1822.

[15] D Gupta and S Choubey. Discrete wavelet transform for image processing. *International Journal of Emerging Technology and Advanced Engineering*, 4(3):598–602, 2015.

[16] E Hildreth and D Marr. Theory of edge detection. *Proceedings of Royal Society of London*, 207(187-217):9, 1980.

[17] J Huang and G Zhou. On-board detection and matching of feature points. *Remote Sensing*, 9(6):601, 2017.

[18] AB Journel and FG Alabert. Focusing on spatial connectivity of extreme-valued attributes: stochastic indicator models of reservoir heterogeneities. *American Association of Petroleum Geologists (AAPG) Bulletin (United States)*, 73:3, 3 1989.

[19] DG Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139, 1951.

[20] A Krizhevsky, I Sutskever, and GE Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[21] DP Laurie. The Roadmaker's algorithm for the Discrete Pulse Transform. *IEEE Transactions on Image Processing*, 20(2):361–371, 2011.

[22] DP Laurie and CH Rohwer. Fast implementation of the Discrete Pulse Transform. In *Proceedings of the International Conference of Numerical Analysis and Applied Mathematics*, pages 15–19. Weinheim, Germany, 2006.

[23] G Li and Y Yu. Visual saliency based on multiscale deep features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5455–5463, 2015.

[24] Y Li, S Wang, Q Tian, and X Ding. A survey of recent advances in visual feature detection. *Neurocomputing*, 149:736–751, 2015.

[25] T Lindeberg. *Scale-Space*. Wiley Online Library, 2008.

[26] T Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1):3–36, 2015.

[27] T Lindeberg. Spatio-temporal scale selection in video data. *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 3–15, 2017.

[28] T Liu, Z Yuan, J Sun, J Wang, N Zheng, X Tang, and HY Shum. Learning to detect a salient object. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):353–367, 2011.

[29] DG Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[30] N Luo, Q Sun, Q Chen, Z Ji, and D Xia. A novel tracking algorithm via feature points matching. *PlOS ONE*, 10(1):e0116315, 2015.

[31] G Mariethoz and J Caers. *Multiple-Point Geostatistics: Stochastic Modeling with Training Images*. John Wiley & Sons, 2014.

[32] G Mariethoz, P Renard, and J Straubhaar. The Direct Sampling method to perform multiple-point geostatistical simulations. *Water Resources Research*, 46(11), 2010.

[33] D Martin, C Fowlkes, D Tal, and J Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on Computer Vision*, volume 2, pages 416–423. IEEE, 2001.

[34] E Meerschman, G Pirot, G Mariethoz, J Straubhaar, M Van Meirvenne, and P Renard. A practical guide to performing multiple-point statistical simulations with the direct sampling algorithm. *Computers & Geosciences*, 52:307–324, 2013.

[35] Z Miao and X Jiang. Interest point detection using rank order log filter. *Pattern Recognition*, 46(11):2890–2901, 2013.

[36] K Mikolajczyk, T Tuytelaars, C Schmid, A Zisserman, J Matas, F Schaffalitzky, T Kadir, and L Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2):43–72, 2005.

[37] BA Olshausen and DJ Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607, 1996.

[38] H Rezaee, G Mariethoz, M Koneshloo, and O Asghari. Multiple-point geostatistical simulation using the bunch-pasting direct sampling method. *Computers & Geosciences*, 54:293–308, 2013.

[39] CH Rohwer. Idempotent one-sided approximation of median smoothers. *Journal of Approximation Theory*, 58(2):151–163, 1989.

[40] CH Rohwer. Variation reduction and LULU-Smoothing. *Quaestiones Mathematicae*, 25(2):163–176, 2002.

[41] CH Rohwer. Fully trend preserving operators. *Quaestiones Mathematicae*, 27(3):217–229, 2004.

[42] CH Rohwer. *Nonlinear Smoothers and Multiresolution Analysis*. Birkhäuser Verlag, 2005.

[43] CH Rohwer and DP Laurie. The Discrete Pulse Transform. *SIAM Journal on Mathematical Analysis*, 38(3):1012–1034, 2006.

[44] CH Rohwer and M Wild. Natural alternatives for one dimensional median filtering. *Quaestiones Mathematicae*, 25(2):135–162, 2002.

[45] CH Rohwer and M Wild. LULU theory, idempotent stack filters, and the mathematics of vision of Marr. *Advances in Imaging and Electron Physics*, 146:57–162, 2007.

[46] E Rublee, V Rabaud, and G Konolige, Kand Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011.

[47] U Rutishauser, D Walther, C Koch, and P Perona. Is bottom-up attention useful for object recognition? In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision*, volume 2, pages II–II. IEEE, 2004.

[48] J Serra. A lattice approach to image segmentation. *Journal of Mathematical Imaging and Vision*, 24(1):83–130, 2006.

[49] CE Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:623–656, 1948.

[50] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] J Stewart. *Calculus: Concepts and Contexts 4th Edition*. Brooks Cole, 2005.

[52] GG Stoltz. Roadmaker's Pavage, pulse reformation framework and image segmentation in the Discrete Pulse Transform. Master's thesis, University of Pretoria, 2014.

[53] SB Strebelle. Conditional simulation of complex geological structures using multiple-point statistics. *Mathematical Geology*, 34(1):1–21, 2002.

[54] SB Strebelle. Multiple-point statistics (MPS) simulation with enhanced computational efficiency, April 7 2009. US Patent 7,516,055.

[55] SB Strebelle. Multiple-point geostatistics: from theory to practice. In *Expanded Abstract Collection from Ninth International Geostatistics Congress. Norwegian Computing Center*, page 65, 2012.

[56] WR Tobler. A computer movie simulating urban growth in the detroit region. *Economic Geography*, 46(sup1):234–240, 1970.

[57] TY Tsou and SH Wu. A robust feature matching method for robot localization in a dynamic indoor environment. In *Technologies and Applications of Artificial Intelligence*, pages 354–365. Springer, 2014.

[58] SJ Van der Walt. *Super-resolution imaging*. PhD thesis, Stellenbosch: University of Stellenbosch, 2010.

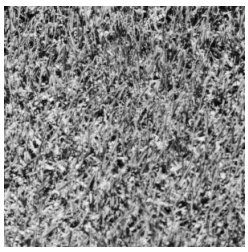
[59] Z Wang, AC Bovik, HR Sheikh, and EP Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[60] R Wu, Y Yu, and W Wang. Scale: Supervised and cascaded laplacian eigenmaps for visual object recognition based on nearest neighbors. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on Computer Vision*, pages 867–874. IEEE, 2013.

[61] D Zeng and M Zhu. Multiscale fully convolutional network for foreground object detection in infrared videos. *IEEE Geoscience and Remote Sensing Letters*, 15(4):617–621, 2018.

[62] R Zhao, W Ouyang, and X Wang. Unsupervised salience learning for person re-identification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3586–3593. IEEE, 2013.

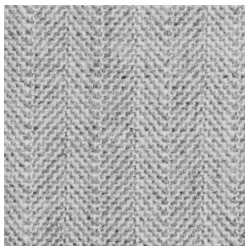
Appendix



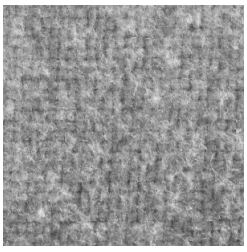
Texture 1: Grass

Texture 2: Bark

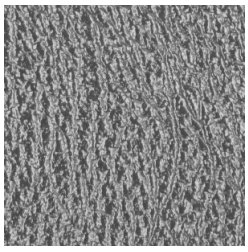
Texture 3: Straw



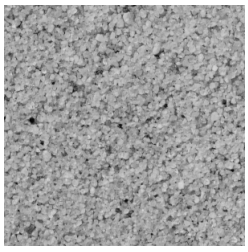
Texture 4: Herringbone weave



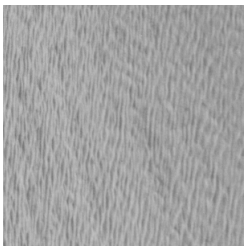
Texture 5: Woolen cloth



Texture 6: Pressed calf leather

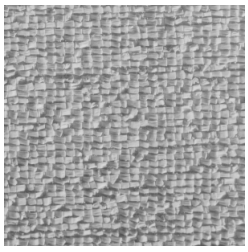


Texture 7: Beach sand

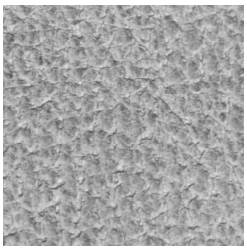


Texture 8: Water

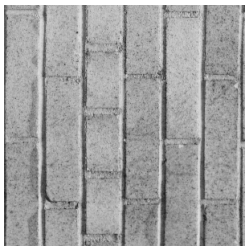
Texture 9: Wood grain



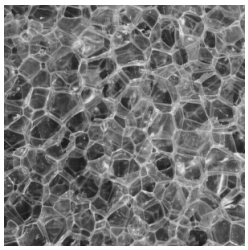
Texture 10: Raffia



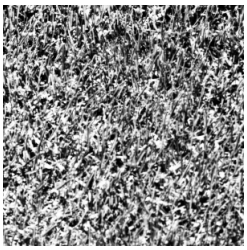
Texture 11: Pigskin



Texture 12: Brick wall



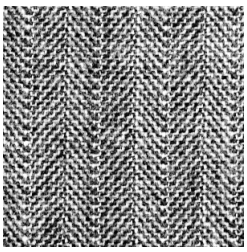
Texture 13: Plastic bubbles



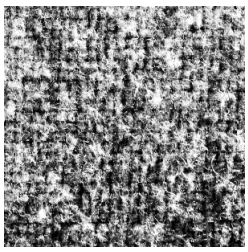
Texture 14: Grass

Texture 15: Bark

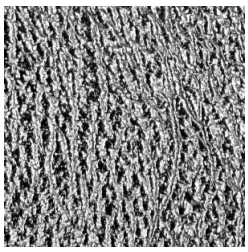
Texture 16: Straw



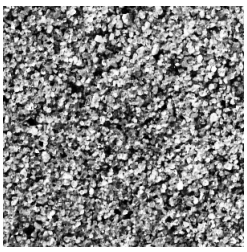
Texture 17: Herringbone weave



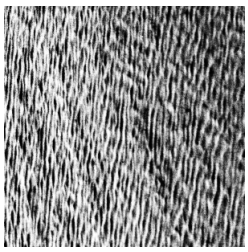
Texture 18: Woolen cloth



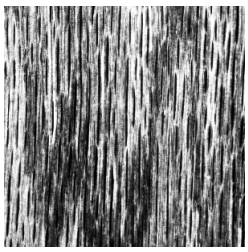
Texture 19: Pressed calf leather



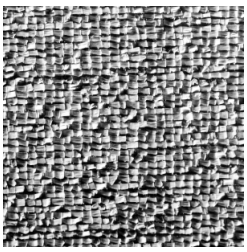
Texture 20: Beach sand



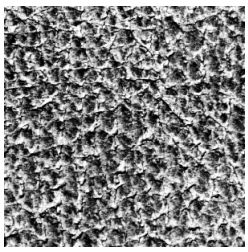
Texture 21: Water



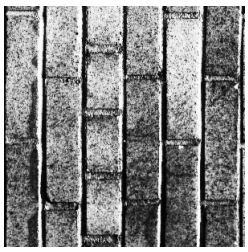
Texture 22: Wood grain



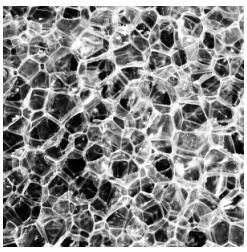
Texture 23: Raffia



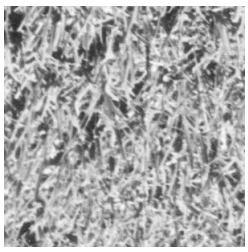
Texture 24: Pigskin



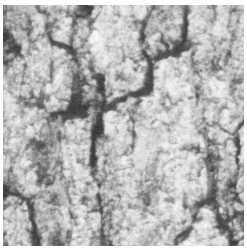
Texture 25: Brick wall



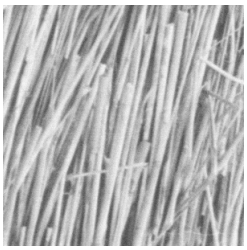
Texture 26: Plastic bubbles



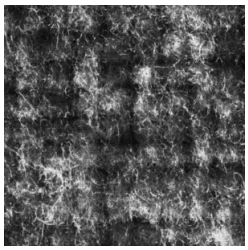
Texture 27: Grass



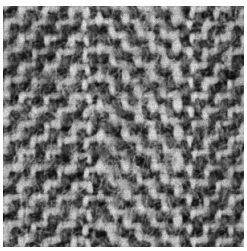
Texture 28: Bark



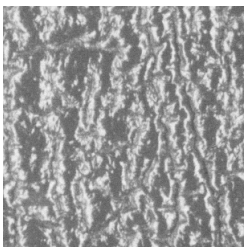
Texture 29: Straw



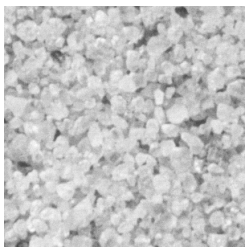
Texture 30: Woolen cloth



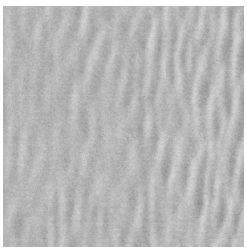
Texture 31: Herringbone weave



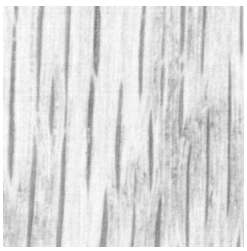
Texture 32: Pressed calf leather



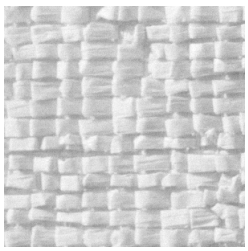
Texture 33: Beach sand



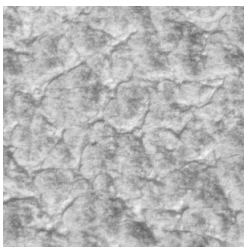
Texture 34: Water



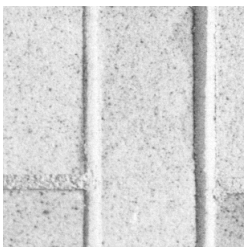
Texture 35: Wood grain



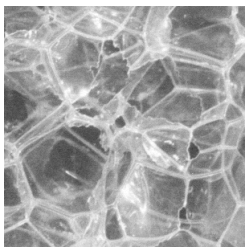
Texture 36: Raffia



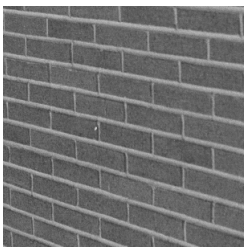
Texture 37: Pigskin



Texture 38: Brick wall

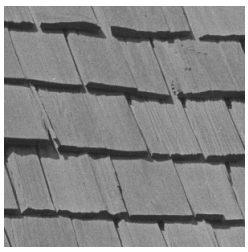


Texture 39: Plastic bubbles

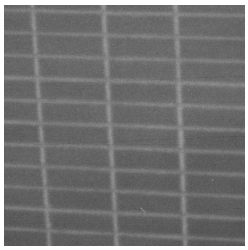


Texture 40: Brick wall

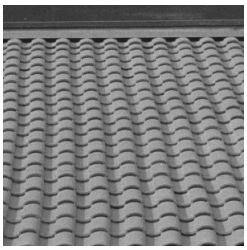
Texture 41: Wood shingle roof



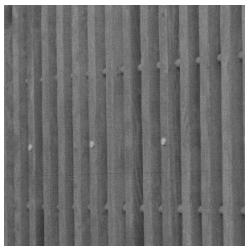
Texture 42: Wood shingle roof



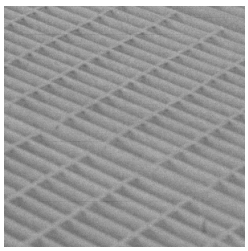
Texture 43: Brick wall



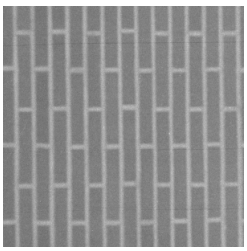
Texture 44: Tile roof



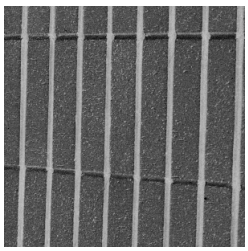
Texture 45: Wood fence



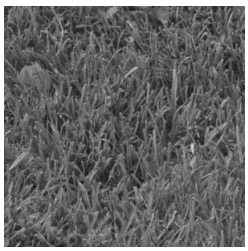
Texture 46: Metal grates



Texture 47: Brick wall



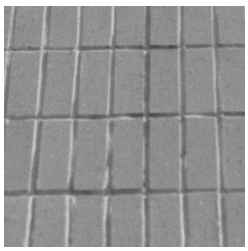
Texture 48: Brick wall



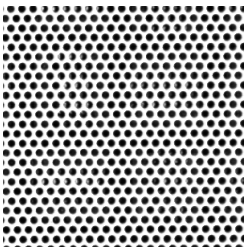
Texture 49: Grass

Texture 50: Sand

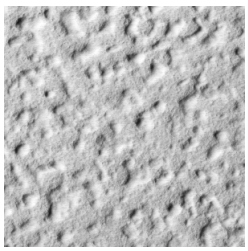
Texture 51: Sand



Texture 52: Brick wall

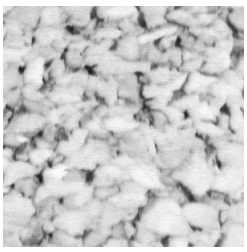


Texture 53: Hexagonal hole array

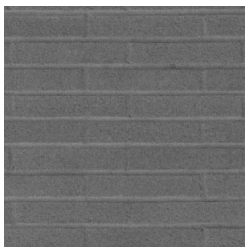


Texture 54: Rough wall

Texture 55: Sand



Texture 56: Gravel



Texture 57: Brick wall

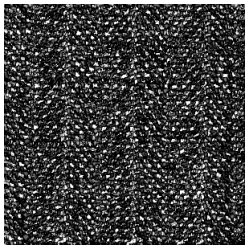
Texture 58: Grass

Table A.1: USC-SIPI texture image database.

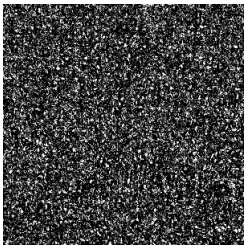
Texture 1: Grass

Texture 2: Bark

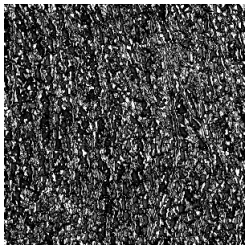
Texture 3: Straw



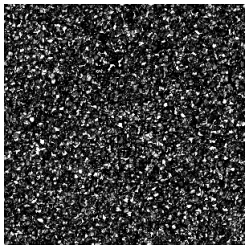
Texture 4: Herringbone weave



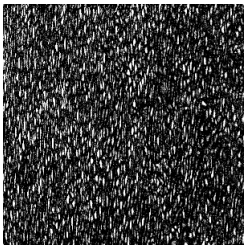
Texture 5: Woolen cloth



Texture 6: Pressed calf leather

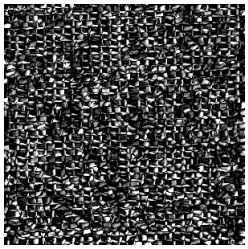


Texture 7: Beach sand

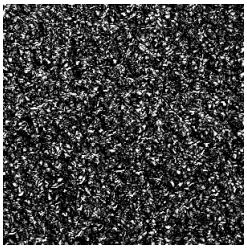


Texture 8: Water

Texture 9: Wood grain



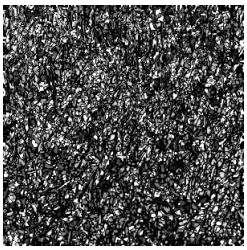
Texture 10: Raffia



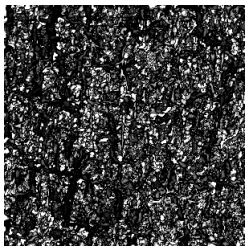
Texture 11: Pigskin

Texture 12: Brick wall

Texture 13: Plastic bubbles

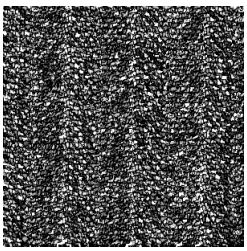


Texture 14: Grass

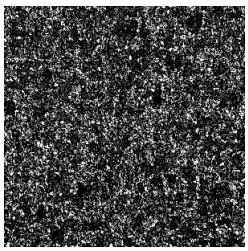


Texture 15: Bark

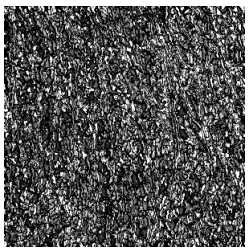
Texture 16: Straw



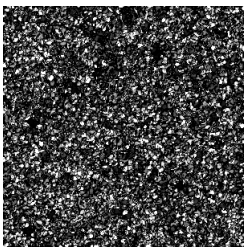
Texture 17: Herringbone weave



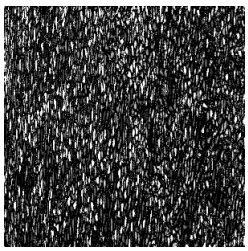
Texture 18: Woolen cloth



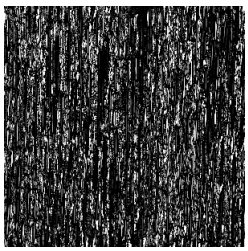
Texture 19: Pressed calf leather



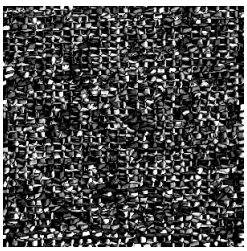
Texture 20: Beach sand



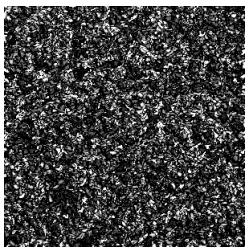
Texture 21: Water



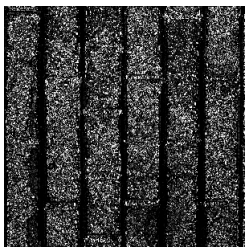
Texture 22: Wood grain



Texture 23: Raffia

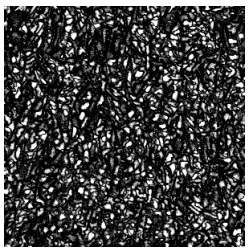


Texture 24: Pigskin

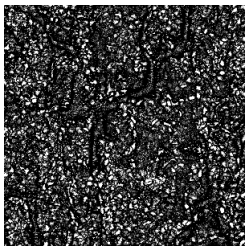


Texture 25: Brick wall

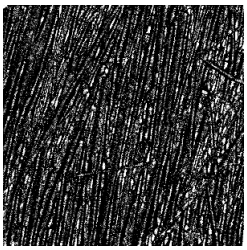
Texture 26: Plastic bubbles



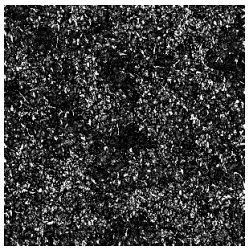
Texture 27: Grass



Texture 28: Bark

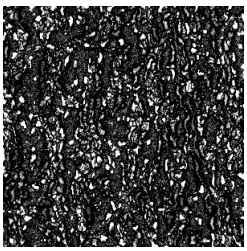


Texture 29: Straw

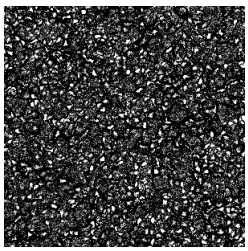


Texture 30: Woolen cloth

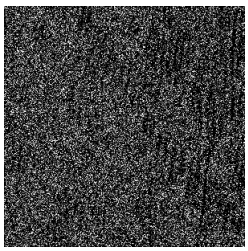
Texture 31: Herringbone weave



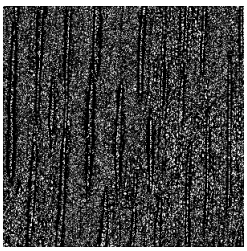
Texture 32: Pressed calf leather



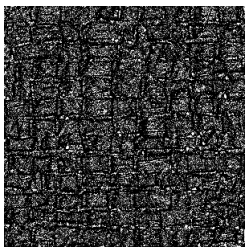
Texture 33: Beach sand



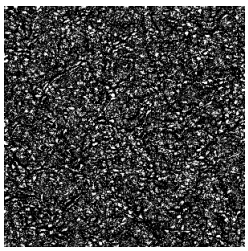
Texture 34: Water



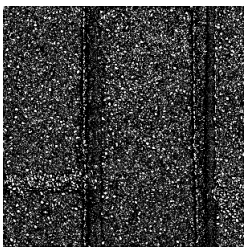
Texture 35: Wood grain



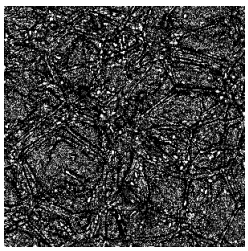
Texture 36: Raffia



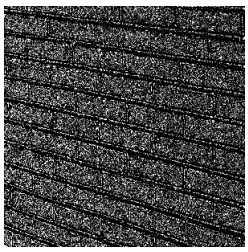
Texture 37: Pigskin



Texture 38: Brick wall



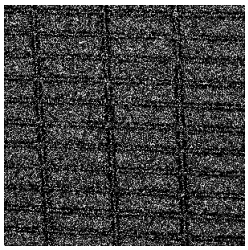
Texture 39: Plastic bubbles



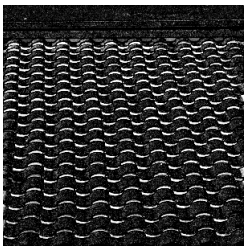
Texture 40: Brick wall

Texture 41: Wood shingle roof

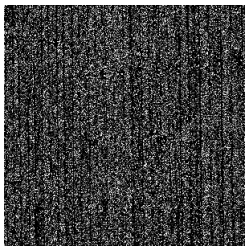
Texture 42: Wood shingle roof



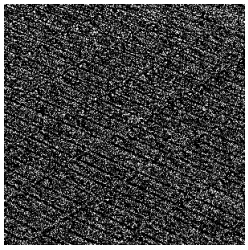
Texture 43: Brick wall



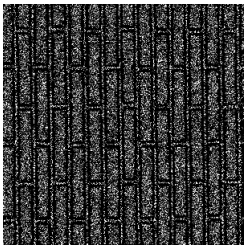
Texture 44: Tile roof



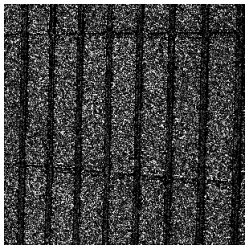
Texture 45: Wood fence



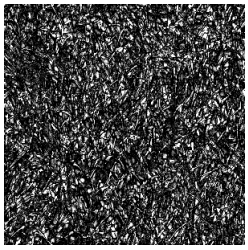
Texture 46: Metal grates



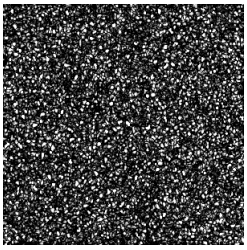
Texture 47: Brick wall



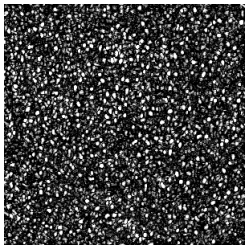
Texture 48: Brick wall



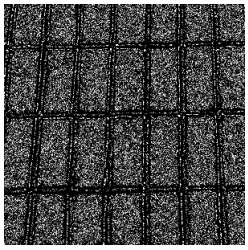
Texture 49: Grass



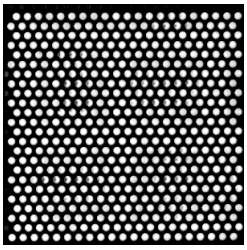
Texture 50: Sand



Texture 51: Sand



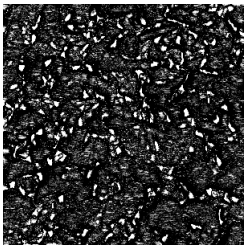
Texture 52: Brick wall



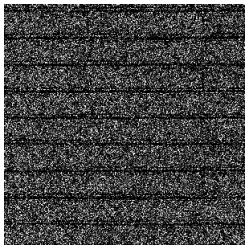
Texture 53: Hexagonal hole array

Texture 54: Rough wall

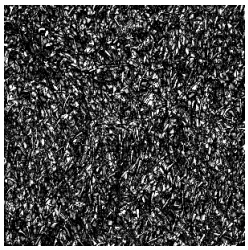
Texture 55: Sand



Texture 56: Gravel



Texture 57: Brick wall



Texture 58: Grass

Table A.2: USC-SIPI texture image database pulse maps.