university of groningen

faculty of science and engineering

Edge Detection with Inhibition for Vectorization

Brian de Jager

university of groningen

faculty of science and engineering

University of Groningen

Edge Detection with Inhibition for Vectorization

Bachelor's Thesis

Under the supervision of
Dr. G. Azzopardi
and
Prof. J. Kosinka

Brian de Jager (s4104579)

October 5, 2023

Contents

1 Abstract

We propose using a computational model of simple cells with push-pull inhibition as a replacement for Canny's edge detector for the purpose of an image vectorization pipeline. The new image vectorization pipeline is based on a pre-existing image vectorization pipeline. We demonstrate that using the new edge detection method in an image vectorization pipeline we can improve the vectorized result, achieve better reliability, and generate simpler meshes. We show this improvement with the use of a handpicked data set of 20 noisy and 20 regular images from the ImageNet Object Localization Challenge - a subset of a widely used data set used for advancing computer vision and deep learning research. The image vectorization pipeline we propose is a contribution to the field of image processing as it provides solid evidence that we can improve the image vectorization process by using different methods of edge detection. As a result, noisy images get vectorized into simpler meshes and will be easier to work with. Additionally, the mean squared error of the vectorized result will be significantly lower.

2 Introduction

2.1 Problem definition and Motivation

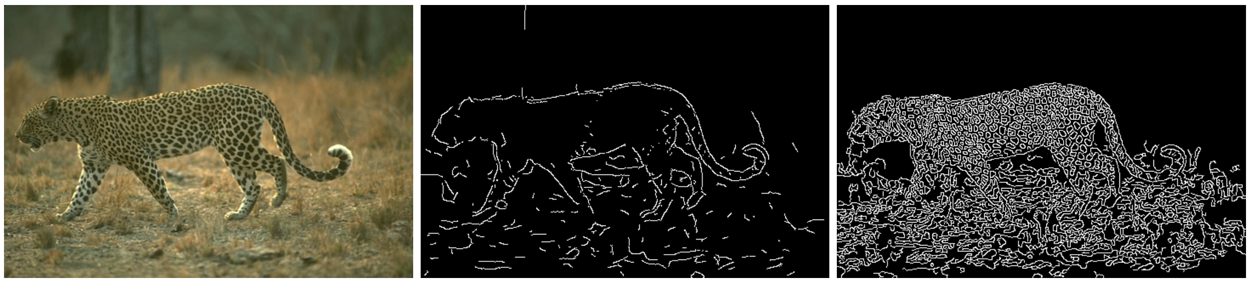


Figure 1: Contour maps of both push-pull CORF and Canny's edge detector

Edge detection is a fundamental problem in image processing. It is the process of constructing a contour map with the most essential edges and object boundaries from a natural image [1]. There are many ways edge detection can be performed, of which a few relevant methods are addressed in Section 3.2. These existing methods have a combination of positives and negatives, and can be used to serve different purposes. In short, one edge detection method can deem edges within a texture as important, where the other method does not. This results in one contour map containing significantly more edges (where all textures are deemed to be important) than the other (where all the textures are not important).

As an example of a textured image, we can have a look at the skin of a leopard. In Figure 1 we show a leopard with a very complicated textured skin. Canny's edge detector, further explained in Section 3.2, emphasizes the dotted skin pattern as important edges (Figure 1 right), whereas push/pull CORF, further explained in Section 3.2, emphasizes only the leopard's outline (Figure 1 middle).

These contour maps can be used in applications to identify objects, but also to serve as a part of the image vectorization pipeline. In essence, an image vectorization pipeline takes as input a raster image and converts it into a vector image. Vector images have significant advantages over raster images, as for example, the file size is independent of the size of the image. To compare how well an edge detection algorithm performs, we can compare it to a ground truth. A method that can be used to construct such a ground truth is mentioned in [2], which uses a hand drawn contour map that shows the edges of objects and shadows, and omits textured backgrounds and irrelevant textures within objects. But to see how well an image vectorization pipeline performs, we can compare the output after rasterization with the input image as the ground truth. Hence, the additional edges can be used to capture the input image with more accuracy and make the image vectorization pipeline perform better, whereas when comparing the edge detectors it would perform worse. For that purpose, it is important to optimize the contour maps to represent the ground truth as much as possible, while also capturing enough textural detail to accurately reconstruct the input image.

2.2 Aims and Objectives

The objective of this thesis is to answer the following research question: to what extent can a CORF-based image vectorization pipeline outperform a Canny-based image vectorization pipeline? This is done by creating an image vectorization pipeline where we can replace the original Canny's edge detector with a new edge detection algorithm called push/pull CORF. The aim is to use this new pipeline with a benchmark data set to compare the performance with the original pipeline. With this, we hope to conclude that by using push/pull CORF as an edge detector we can improve the performance on noisy images.

2.3 Scope

In this thesis, we want to set up a test to investigate the possibility of altering the edge detector of the Canny-based image vectorization pipeline to increase its performance. As it is yet unknown if it would increase the performance, and because push/pull CORF is created in Matlab and the pipeline is in C++, we look at it as separate entities. Therefore, no integration of the edge detector is necessary, which leaves space for future work either to include push/pull CORF in the pipeline or investigate different edge detectors for image vectorization. Hence, supplying additional contour maps to the pipeline instead of using the default edge detector is sufficient at this time. As a major benefit of this, the pipeline can be tested with a whole set of different edge detectors in the future.

Besides that, altering the pipeline in any other way to tackle issues that are more prevalent when using a new edge detector falls outside of the scope of this project. As the vectorizer is built around Canny's edge detector, there might be some implementations that are specifically included to increase the performance of the original version. The pipeline could be altered to resolve some issues, but this falls outside of the current scope. In Section 4.3 we dive into the implications of the present issues.

2.4 Overview of the proposed solution

We propose an altered version of the pipeline where we can add contour maps as input. These contour maps will replace the output of the present edge detector. The new pipeline is used to test whether push-pull CORF can increase the pipeline's performance.

Firstly we alter the pipeline to be able to use the contour maps as input besides the input image. Secondly, we investigate the initial hypothesis that heavily textured input images result in a significantly better vectorized result. This is done by testing the robustness of the new pipeline to white Gaussian noise at various variance levels and comparing the performance between the two pipelines. Thirdly, we analyze the current drawbacks of the method we use to compare the results. As there are still vectorization mistakes within the resulting images, it is very hard to compare them fairly. Hence, we try to explain in what ways the current implementation falls short, as fixing these are not within the scope of this project but argue why it is still the right way to compare them. Fourthly, we compare the output of the two pipelines using a benchmark data set with 20 noisy and 20 regular images, by comparing the mean squared error.

2.5 Structure of the report

In Chapter 3 we look at the related work and explain common terms used in the related work. In Chapter 4 we show the layout of the CORF-based pipeline, how it differs from the Canny-based pipeline, compare the robustness of the two pipelines, and show the present issues with both of the pipelines. In Chapter 5 we show the design of the experiment, together with a summary of the produced results. In Chapter 6 we look at the produced results to compare the two pipelines. Finally, in Chapter 7 we draw a conclusion based on the comparison and try to answer the research question.

3 Background and Literature Review

There are various different ways to construct a contour map from an image. We will look at a set of existing edge detectors, explain how a subset of these edge detectors work in-depth, and analyze what kind of edge detectors are used in existing image vectorization pipelines. To better understand the technical terms in the literature review we start with some background information regarding edge detection and image vectorization.

3.1 Background

Throughout this thesis, various terms are used that are commonly used in this field of study. This background review describes the basic definitions of these used terms and shows how they relate to this thesis.

Edge detection Edge detection is used in many image processing applications, and can be described as capturing geometrical characteristics of objects in an image [3]. The resulting image is a contour map with the most essential object boundaries (also called edges) from a natural image [1]. Depending on the type of edge detection algorithm and its parameters, it can filter out the most important edges or try to approximate all edges from the image. As the human eye cannot perceive the level of detail a computer can, filtering out less important edges can go unnoticed.

To evaluate what edge detection algorithm is best suited for the job, the algorithms are judged by common criteria such as low error rate and good localization [4]. The low error rate is self-explanatory: important edges should be detected by the edge detection algorithm. Besides that, localization is important because wrongly placed edges can lead to side effects when they are used as input for other applications. As an example, when an edge does not properly bind the real object's shape, meaning colors from the original image fall outside of the detected edge due to poor localization, a colour bleeding effect might occur when using this contour map for image vectorization. Hence, the resulting edge should be as close as possible to the actual edge in the input image.

Image vectorization The process of image vectorization is converting a raster image (an image built of pixels) to a vector image. A vector image is a collection of geometric primitives or more complex shapes [5]. These vector images have great advantages, such as the possibility to resize them without losing definition and saving memory space for larger-sized images. As raster images are saved per pixel, the storage size scales linearly with the size of the image. Vector images on the other hand are a collection of shapes, and so these shapes can be scaled to any size without requiring more memory space.

Bézier curves A Bézier curve is a way to describe a line element using a mathematical formula. A Bézier curve has a set of control points, of which the amount of control points depends on the degree of the Bézier curve. As the pipeline used in this research uses cubic Bézier curves, it is based on 4 control points. With these control points and the basic cubic Bézier curve formula we can create a curve. Hence, we can use them to describe all edges from the contour map that is generated using either Canny or push-pull CORF.

Mean squared error The mean squared error is calculated per pixel. It tries to approximate the overall difference of the resulting image compared to the ground truth. For each of the three colour channels at each pixel, we compare its value to the corresponding pixel in the ground truth. At each pixel for each colour channel, we subtract the pixel values and square them. We have to square the difference to make sure there are no negative differences, and to make big differences account for more than small differences. Afterward, we divide the squared error (the sum of all pixel errors) by the number of pixels, resulting in the mean squared error over the whole image.

Compression ratio The compression ratio is calculated based on the input file size and the output file size. By dividing the output file size with the input file size, we get the compression ratio. Hence, if the output file size is larger than the input file size, we get a compression ratio greater than 1.

3.2 Literature review

As image processing is a well known field of study, a lot of research has been done in this field. A subset of the research is done in trying to automate the process of image vectorization, as manually tracing images might need significant amounts of labor time as well as expertise [6]. To reduce the amount of labor time and allow people with less expertise to vectorize images, we can try to construct an image vectorization pipeline. Important is the quality of the resulting image, so it is desired to capture the most important edges, while also keeping the file size to a minimum.

There are three categories of image vectorization programs: Interactive image vectorization, semi-automatic image vectorization, and automatic image vectorization [7], varying in the degree of human interaction with the system. For this research we focus on the category of automatic image vectorization, using an image vectorization pipeline. This means a system where a raster image is used as input and receives a vector image as output without needing additional human interaction.

Current image vectorization pipelines In the category of automatic image vectorization, we can find different image vectorization pipelines, either with an explicit edge detection algorithm or one integrated into the system. For example, [8] describes a progressive learning pipeline that fits Bézier paths to connected areas with uniform-filled colour instead of using a contour map. Another example that is used for a subset of images, namely hand-drawn cartoon images, is a process called skeletonization that can be used with the use of constrained Delaunay triangulation [9]. Because hand-drawn images are line drawn, the lines are the skeleton used for the triangulation process, and so no additional edge detection needs to be used.

In the explicit edge detection algorithms category, we mostly find vectorization pipelines that use either the Canny edge detector or the Sobel operator. [10] makes use of Canny's edge detector with additional temporal averaging for noise reduction in the pipeline. This method of edge detection is also found in the pipeline mentioned in [5].

Commonly use edge detectors As can be seen, all mentioned image vectorization pipelines make use of some sort of edge detection. In the category of explicit edge detection, Canny's edge detector is very commonly used. But one can use other methods to detect edges from a raster image such as using specific layers of a deep convolutional neural network using richer convolutional features [1], a computational model of simple cells with push-pull inhibition [2], and many more not named further.

Push/pull CORF Push-pull CORF is a computational model that makes use of a property that is found in many real simple cells. Simple cells can also be found in the visual cortex. This is the part of the brain that processes visual information and is composed of several layers of neurons that perform various operations.

These layers are built with a combination of simple, complex, and hyper-complex cells [11]. The simple cells are good at detecting the contour of an object, which is exactly the goal of edge detection. Push-pull CORF uses a combination of receptive fields (CORF). A CORF model takes as input the response of a group of lateral geniculate nucleus cells. These responses are combined with the use of a weighted geometrical mean. Push-pull CORF takes as input the responses of two CORF models: one with preferred polarity and one with opposite polarity. If two receptive fields are of opposite polarity, it means that one of the two receptive fields is center-on and the other one is center-off. To explain the difference, for center-on receptive fields the more light hits the center but not the surrounding region the higher the action potential of that neuron will be. For center-off receptive fields, the less light hits the center and the more light hits the surrounding region the higher the action potential of that neuron will be. So, in center-on receptive fields, the center is excitatory and the surrounding region is inhibitory. Similarly, in center-off receptive fields, the center is inhibitory and the surrounding region is excitatory [12]. By combining the output of both receptive fields it computes its response based on the difference of these CORF model's responses. The push-pull CORF model has a good signal-to-noise ratio (SNR), significant separability of spatial frequency and orientation, and contrast-dependent changes in spatial frequency tuning, which is also found in real simple cells [2].

Push-pull CORF can be adjusted with a set of 4 parameters: the scale factor σ, high threshold value ζ, inhibition factor k, and β. The amount of push-pull inhibition is decided by the inhibition factor k and β. With the correct setting, it can represent phenomena that are similar to behavior found in real simple cells. The value β affects how strongly the model responds to the response of preferred frequency. Together with a scale parameter σ that enhances differences in the distribution of the model's result a response is obtained. This is followed by a thresholding step called hysteresis to generate a binary contour map. Thresholding is done with 2 values, namely a high and a low threshold value. The low threshold value is 0.5 times the high threshold value [2]. Throughout this thesis, the settings of CORF will be depicted as (σ, β, k, ζ), or as (σ, β, k) when a test goes over a range of ζ values.

Canny's edge detector Canny's edge detector focuses on a low error rate and proper localization by defining a mathematical form. Canny's edge detector is composed of 3 steps: Detection, localization, and thresholding. To detect edges the image is slightly blurred using Gaussian smoothing. This step gets rid of the first layer of noise. This is necessary for the following step because the Sobel gradient operator is very prone to noise. With the use of the Sobel gradient operator, it creates a map of magnitudes in the x and y directions with which it can calculate the orientation of the gradient and the total response at each pixel. It uses non-maximum suppression (also called thinning) for the localization step. This ensures that each edge is 1 pixel wide at the place where the magnitude is maximum keeping in mind the orientation of the magnitude. Finally, the thresholding step makes use of hysteresis thresholding. This is a method based on two thresholds, high and low, and selects all edges that are above the high threshold, and selects all edges that are above the low threshold and connected to an edge that is above the high threshold [4].

To further enhance finding the most important features, Canny's operator can be used with edge scales. Edge scales are a method to add blur a set amount of times (the number of edge scales). This is used to remove less prevalent edges and unnecessary noise from the image while important well-defined edges stay. In other words, if an edge does not persist over a set amount of edge scales, the edge will not be used as a hard edge.

The results of Canny's edge detector can be influenced by the following parameters: a high threshold, a low threshold, and the number of edge scales.

Current results on comparing contour maps Specifically interesting for this research is using a computational model of simple cells with push-pull inhibition [2] and Canny's edge detector [4]. Push-pull CORF outperforms the Canny edge detector with statistical significance [2] when comparing the results with the use of Matthew's correlation coefficient. This paper does not mention if any edge scales are used, which can have a significant impact on the result. In the pipeline structure mentioned before edge scales are used, and so it is not certain that CORF with push/pull inhibition will outperform Canny's edge detector with edge scales. Additionally, [2] uses a method described in [13] to deal with inaccurate localization by saying a detected pixel of an edge is correct if there is a corresponding pixel in a 5x5 neighbourhood in the ground truth. Therefore, it is difficult to say how these contour maps would compare if the exact pixel location would matter.

Existing literature gap As Canny's edge detector is very popular, it is commonly used in existing applications. Although this is commonly used, it might not be the best edge detector to use or not the best to use in certain scenarios. Therefore, it is uncertain if replacing Canny's edge detector with another edge detection algorithm such as CORF with push/pull inhibition will improve an image vectorization pipeline's performance. Hence, it is worth investigating if there is a class of images that gets a better vectorized result using a different edge detector, or if one edge detector is more resilient against some kind of noise.

4 Method

In this thesis, we replace Canny's edge detection algorithm with a computational model of simple cells with push-pull inhibition [2] to detect edges in the image vectorization pipeline [5]. This computational model of simple cells is implemented in MATLAB and can be found at [14].

4.1 Overview

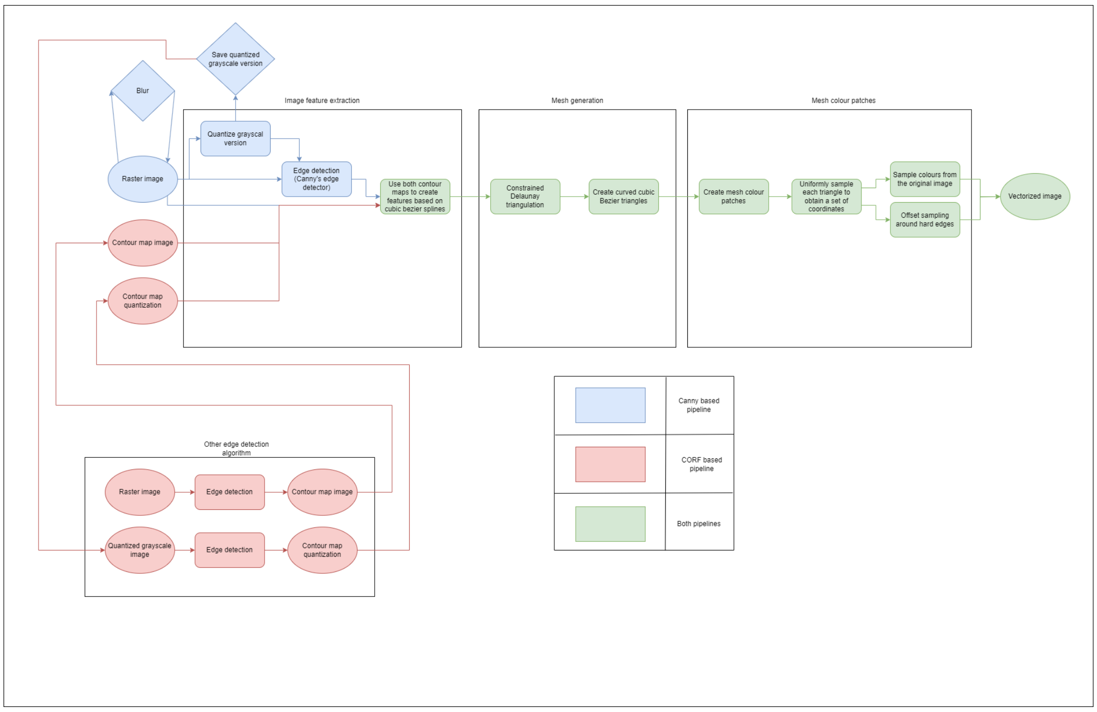

Save quantized grayscale version
Blur
Image feature extraction
Mesh generation
Mesh colour patches
Quantize grayscal version
Raster image
Edge detection (Canny's edge detector)
Use both contour maps to create features based on cubic bezier splines
Constrained Delaunay triangulation
Create curved cubic Bezier triangles
Create mesh colour patches
Uniformly sample each triangle to obtain a set of coordinates
Sample colours from the original image
Offset sampling around hard edges
Vectorized image
Contour map image
Contour map quantization
Other edge detection algorithm
Raster image
Edge detection
Contour map image
Quantized grayscale image
Edge detection
Contour map quantization
Canny based pipeline
CORF based pipeline
Both pipelines

Figure 2: CORF and Canny-based pipeline flowchart

As can be observed in the flowchart in Figure 2, there are 3 tracks: The Canny-based pipeline (blue), the CORF-based pipeline (red), and the parts where they overlap (green). Additionally, squares represent a necessary action of the pipeline, diamonds represent optional actions and circles represent a file.

As expected, the parts differ where the edge detection takes place. We use edge detection two times: on the input image and a grayscale quantized version of the input image. We use the grayscale image to create soft edges around colour changes to make sure we capture noticeable colour differences. Without this additional grayscale image step, we would see noticeable triangles in the result. To run the new edge detector we first have to run the default pipeline to save the quantized grayscale version. We run the new edge detector on both the raster image and the quantized grayscale image to obtain two contour maps. These contour maps serve as an input to the new pipeline. In the vectorizer settings, not displayed in this flow chart, one can easily switch between the blue track and the red track; do note that it is essential to supply both of the contour maps when running the red track.

An additional change done on the default pipeline to make a fair comparison is that we use the same edge detection settings for both the original image and the quantized grayscale version.

Canny-based pipeline The Canny-based pipeline uses Canny's edge detector integrated into the pipeline. Part of Canny's edge detector uses a Gaussian smoothing blur, which is necessary as the Sobel gradient operator of the edge detector is prone to noise. Additionally, as part of the edge detection we use 3 edge scales, meaning the input image is blurred 2 additional times. For an edge to be in the final result it should remain in multiple edge scales.

CORF-based pipeline The CORF-based pipeline is reliant on 2 additional input files, namely the contour map image and contour map quantization. These are created based on the input raster image and the saved quantized grayscale version of the input raster image. To have a fair comparison, the saved quantized grayscale image is the same as used in the Canny-based pipeline, and so it is created based on the input image with one layer of blur.

4.2 Robustness

To test the robustness of both the Canny-based pipeline and the CORF-based pipeline we introduce white Gaussian noise at different levels of variance and compare both the MSE and the compression ratio. For image vectorization, the eventual file size is very important. The simpler the image is described by vectors, the easier it is to edit the vectorized result afterward. Because of these requirements, the vectorized image's file size is an important criterion in selecting the best vectorized result similarly to the quality of the vectorized result.

As CORF is good at ignoring noise when generating a contour map compared to Canny's edge detector, increasing the variance of the noise (in other words making the noise more severe) the CORF edge detector should create fewer edges, and so lower the complexity of the result (and therefore lower the eventual file size). Additionally, it should retain as much of the underlying shape as possible.

White Gaussian noise The add white Gaussian noise to an image, we use the Matlab function imnoise() with the parameters 'Gaussian', mean = 0, variance = x where $x \in \{0.05, 0.1, ..., 0.5\}$.

Canny's current limit Because of the introduced edge scales, Canny's edge detector is good at ignoring low-variance noise. This can easily be explained by looking at what edge scales do: they blur the image and see if the edges persist. The lower the variance the more easily such noise is blurred out as the colour difference is less prevalent.

To approach the limit of using Canny's edge detector for vectorization we keep increasing the variance until the vectorization pipeline cannot generate a result. At this point, Canny's edge detector generates too many edges, and a local maximum is found. To decrease the number of edges we can increase the low and high threshold values. At some point, the thresholds reduce the number of edges by such a significant amount that we reach a state where the pipeline runs again. By repeating this process we reach a state where no edge is strong enough to be considered a soft or hard edge. Hence, this results in the smallest file size possible together with the highest possible MSE.

During the process of increasing the threshold, we go over the maximum achievable result (a balance between a low file size and a low MSE).

CORF's current limit Both the CORF and Canny's edge detector make use of thresholds for hysteric thresholding, and so both can change these settings. For CORF we also can increase the threshold until no soft and hard edges remain, and also go over the most optimal setting per image.

Experimental Setup During previous experiments, CORF with the settings $(1,2,1.8,0.007)$ and $(1,4,4,0.007)$ produced the best results over a set of tests. Therefore, we test CORF with two pipeline setups $(1,2,1.8,x)$ and $(1,4,4,x)$ where $x \in \{0.006, 0.009, ..., 0.027\}$.

For Canny we adjust the low and high threshold with the following sets of values: [(15, 100), (30, 200), (45,300), (60, 400), (75,500), (90, 600)]. These values are based on testing with noisy images, from which we concluded that a low threshold of 90 and a high threshold of 600 would result in no soft and hard edges. Additionally, we changed the pipeline with Canny to use the same settings for edge detection and for creating bands (as we also do this for push-pull CORF).

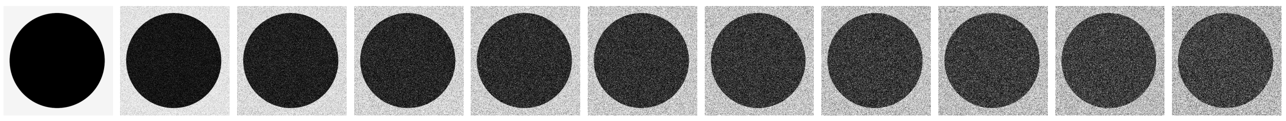


Figure 3: White Gaussian noise stimulus at 11 variance levels

The experiment starts with a black disk on a white square of size 500x500. To analyze at which point CORF will outperform Canny on noise images, we keep adding white Gaussian noise to this black disk. For this, we use the formula *imnoise*() mentioned in Section 4.2 resulting in 11 test images, shown in Figure 3.

Findings With the previously described experiment, we get a set of data points per variance level for 3 different versions of the pipeline. To analyze the robustness, meaning how well the 3 versions can handle the added noise, we compare the best MSE value at each variance level and their compression ratio, which can be observed in Figure 4. Comparing the minimum MSE values with their compression ratio shows us that the pipelines are very similar on low variance noise between the values 0 and 0.2. This can be explained by the edge scales that filter out much of the low-variance noise. From this point onwards the difference in MSE is small but the difference in compression ratio is significant. Afterward, Canny cannot produce results close to both of the CORF versions and the lines of the MSE graph diverge: the Canny-based pipeline produced significantly worse results compared to both of the CORF-based pipelines. What can also be observed is the fact that the compression ratio fluctuates a lot between tests, and is therefore more difficult to analyze. Hence, we can conclude that using the MSE is the best way to judge a pipeline's performance.

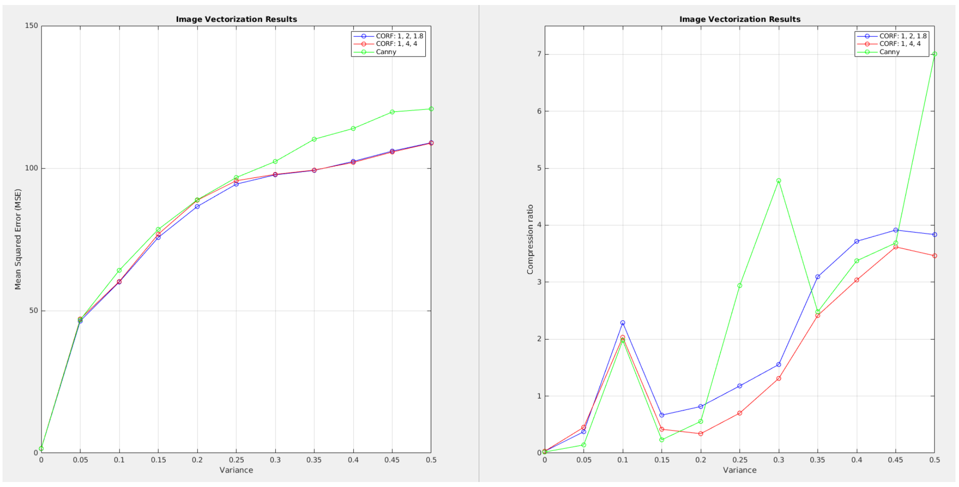


Figure 4: Best MSE and compression ratio value per variance level

To see how the overall distribution of the results is influenced by noise, we have created a plot with all MSE values at all variance levels. As can be seen in Figure 5 the spread between results at a variance level gets larger as variance increases. The spread can be caused by a larger difference in MSE values between the 3 pipelines or by a larger difference between the threshold values of a pipeline. In either case, the spread between results is a good indication of the level of variance at hand.

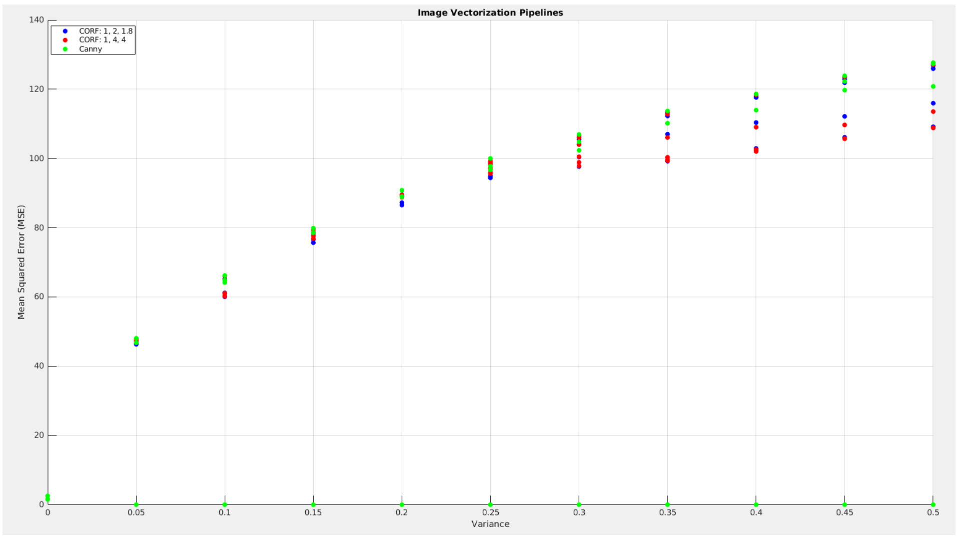


Figure 5: All MSE values per variance level

4.3 MSE value perspective

The mean squared error might not be the best measure to compare the result of the image vectorization pipelines. This is because the quality of the vectorized result also differs greatly without adding noise. The paragraphs below show 3 cases where the vectorization pipelines show unpredictable behavior. This is not to say that the results from the paper are deemed to be inaccurate, more to say that reducing these effects in future work can improve the quality of the noise analysis.

As mentioned in Section 3.2, offset sampling removes a big part of the colour-bleeding effect. But in some cases, there is some active colour bleeding left. We have identified 2 cases where colour bleeding occurs: the edge detector's edge does not capture the entire colour discontinuity and the cubic Bézier spline fitting error margin is too high. Additionally, a place where the mesh is too tightly packed results in undefined lines. This is most likely caused by how imgui on Linux handles graphics. We have decided that fixing these aforementioned issues falls outside of the scope of this project. Therefore, the cases further explained in this section affect the final results and cause the MSE to be higher.

Insufficient localization In some cases, the edge detector's edge fails to capture all of the outline pixels. This results in a pixel outside of an edge, where its colour is similar to the pixels inside the edge.

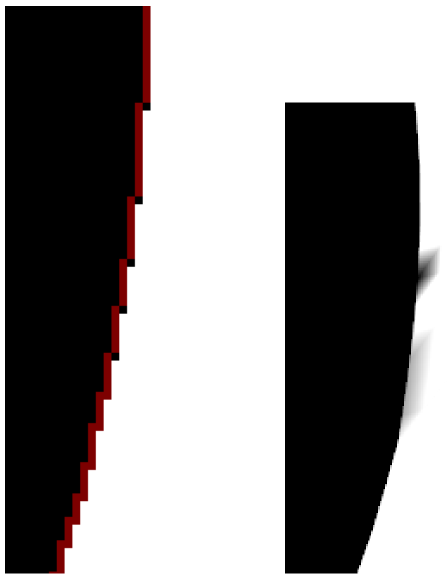


Figure 6: Colour bleeding example with the detected edge

Figure 6 shows how the detected edge (shown in red, left) over the original image (shown in black, left) can cause colour bleeding (shown in right). As can be seen there are a few black pixels outside of the detected edge. This result gives a set of triangles the wrong colour, as can be observed in Figure 6 (right).

If a pixel of a colour discontinuity sits on the wrong side of the edge, it does not necessarily mean that colour bleeding will occur there. This is also dependent on where we place the mesh colour patches.

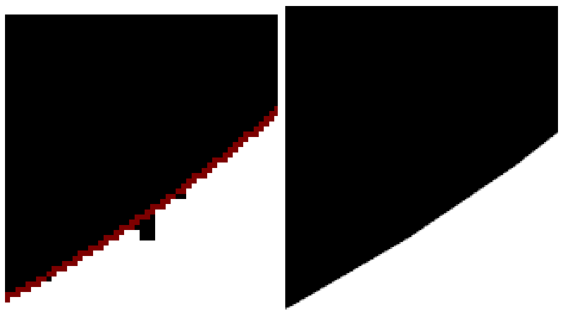


Figure 7: Pixels outside of the detected edge without colour bleed

For a colour bleeding to occur the area has to be sampled. To show an example of a case where we have black pixels outside of the edge, but will not be shown as colour bleeds we can look at Figure 7. In this example, we added a few black pixels to the original image without creating a new edge, which can be observed in Figure 7 (left). As these additional black pixels are on an area that does not get sampled, they do not show as colour bleeds, as can be observed in Figure 7 (right).

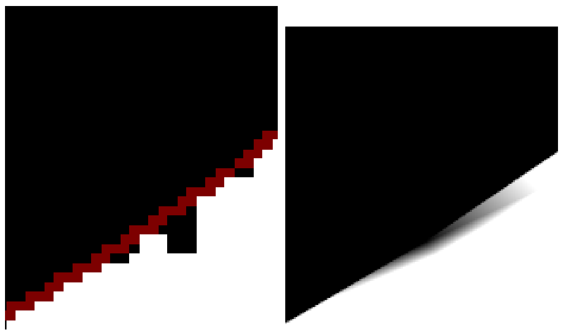


Figure 8: Pixels outside of the detected edge with colour bleed

To show that the pixel position matters for a colour bleed to occur, we have created another example that can be seen in Figure 8. Compared to Figure 7 we have added 3 additional black pixels to the left of the already altered pixels, on a position where the colour gets sampled. With this addition, a colour bleeding effect can be observed in Figure 8 (right).

Cubic Bézier spline fitting error margin The colour bleed can also be caused by the feature extraction step. It might be the case that the curve that goes through the edges does not represent the edge from the contour map well enough, so it leaves out some pixels. In these cases, the Bézier spline fitting error margin is too high. Therefore, lowering the edge fitting max distance can reduce colour bleeding, as this makes sure the hard edge follows the edge from the contour map more closely. This will be at the cost of performance, as curves must be split up more often to decrease the edge fitting max distance.

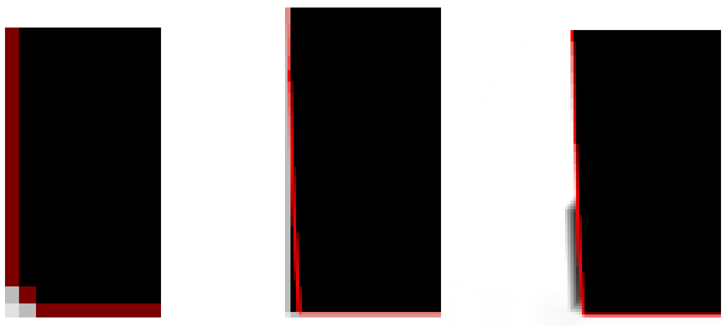


Figure 9: Colour bleed example with the detected edge and resulting feature

As can be seen in Figure 9 a colour bleed occurs (right) although the edge captures all black pixels (left). Because the corner is not a 90-degree angle, the cubic Bézier spline that runs across the bottom left corner cuts off the corner too much (middle). This curve does not get split up in 2 curves, because the edge fitting distance error is lower than the edge fitting max distance.

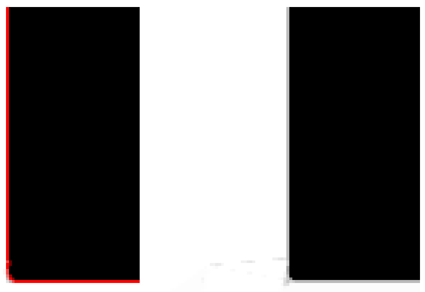


Figure 10: Colour bleed example with the resulting feature at lower edge fitting max distance

If we lower the edge fitting max distance we get the result in Figure 10. As can be seen (left) the hard edge better captures the shape, and so the result (right) does not show the big black colour bleed as in the previous case. Additionally, if we zoom in on Figure 10 we can still notice a minor colour bleed. As described before, this can be explained by the fact that the edge in Figure 9 (left) does not capture all coloured pixels together with the fact that the colour gets sampled on the corner.

Undefined lines In some cases, the vectorized result shows a set of undefined lines as can be seen in Figure 11 (left). For simple shapes, this occurs in places where the edge is discontinuous Figure 11 (middle). As can be seen on Figure 11 (right), the mesh gets complex around this area. Because the pipeline recognizes these meshes as high-resolution triangles, these mesh triangles get split up in a lot of patches.

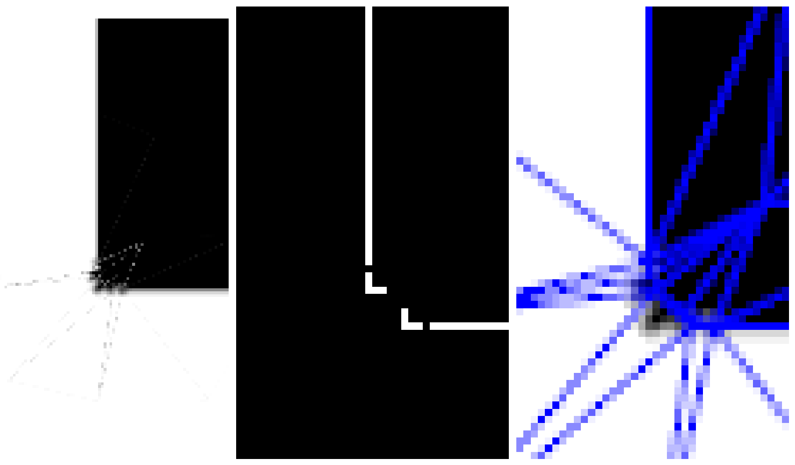


Figure 11: Undefined lines when the detected edge is discontinuous

In Figure 12 we can see that the areas of high-density patches overlap with the lines forming in the result. Hence, the hypothesis is that we cannot render these high-density patches. Zooming in on this area confirms the hypothesis, as now the patches are not as high density, and the undefined lines disappear.

Findings To conclude, both pipeline versions have to deal with colour bleeding and undefined lines. It is very difficult to say how CORF and Canny compare in these challenges when the input imagery gets more complex. What we can say is that for simple shapes it seems that CORF is having more difficulties trying to capture the full colour discontinuity on an edge, and more colour bleeding occurs. Hence, we can assume that for more complex images, the CORF edge detector will have more complications than Canny edge detector and that their pipelines will get influenced accordingly. With this in mind, if a CORF-based pipeline can still outperform a Canny-based pipeline when facing these challenges, it might be a potent candidate when the colour bleeding effect would get solved.

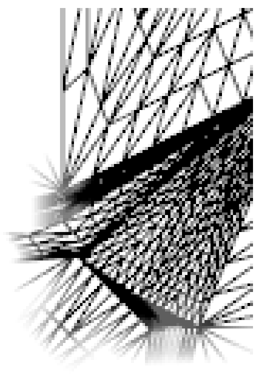


Figure 12: Patches where an undefined line occurs

5 Experiments and results

In this section, we show how we set up the experiments and present the findings of these experiments.

5.1 Dataset

The benchmark dataset used for this research is part of a large ImageNet dataset [15] and is also used for the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Hence, it is well known to be used in image recognition tasks, and to test the quality of these tasks.

We have handpicked 20 perceived noisy images and 20 perceived regular image from the test data set. These images vary from plants, objects, animals, and are chosen to vary between all of these categories. Hence, results are not subjective to a category - although it could be that a category produces better results than others. The noisy images all contain some sort of noise: these range from busy textures, a complex set of lines, or any other kind of noise. The amount of noise varies among the chosen images, and the results might show that difference as well.

5.2 Design of experiments

As mentioned previously we have 2 variables: MSE and Compression ratio, which we both try to minimize. In Chapter 4 we have concluded that the most important measure of the two is the MSE which is used to compare the results. We use the 2 versions of the CORF-based pipeline and compare this to 1 version of the Canny pipeline as described in Section 4.2. Like when testing the robustness in Section 4.2, we gather data using these pipelines with the aforementioned threshold intervals.

The results display the lowest MSE between threshold values and compare them against the two other pipeline versions. This is plotted for both of the datasets separately, after which we can compare between datasets which pipeline performs best for each test (noisy and regular).

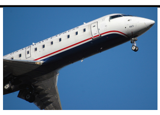
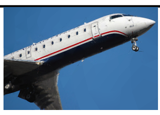
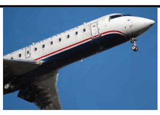
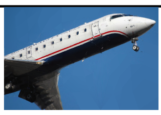
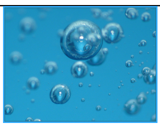
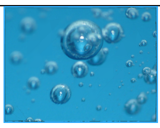
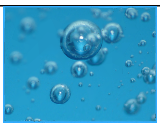
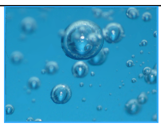
As the data set is handpicked, the images from the noise test can be in fact regular and vice versa. To cross reference the handpicked images we scatter plot all of the MSEs of all tests, and check the variance between results. As previously observed, a large spread of MSE values between tests indicates a large variance in the image. Therefore, we expect the noise test to have a lot of spread between the results and the regular test to have significantly less. Additionally, as noisy images are more complex than regular images, we expect the noisy test to have a higher average MSE than the regular test.

5.3 Results

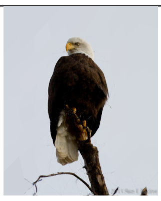
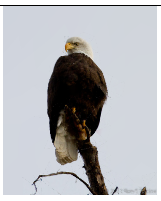
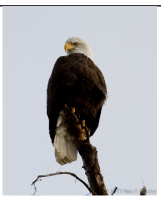
To illustrate the difference between the vectorized results, we create a table with the ground truth and all the relevant output images. The first column represents the ground truth. This is used, together with the output image, to calculate the MSE. The other 3 columns contain the resulting vectorized image after rasterization with the lowest MSE value over the set of threshold values. Hence, for each column, we only show the values of σ, β, and k as ζ might differ for each row. This rasterization process is done by taking the pixel value from the frame buffer for each pixel.

5.3.1 Regular test

In Table 1 we show the 20 regular image ground truths and the corresponding best outputs for each pipeline over the set of threshold values.

Regular test images				
Test	**Ground truth**	**Canny**	**CORF (1, 2, 1.8)**	**CORF (1, 4, 4)**
1		MSE: 7.628 	MSE: 8.204 	MSE: 8.093 
2		MSE: 11.748	MSE: 12.084	MSE: 12.071
3		MSE: 6.39	MSE: 6.718	MSE: 6.634
4		MSE: 7.881	MSE: 8.17	MSE: 7.74
5		MSE: 6.572	MSE: 6.995	MSE: 7.264
6		MSE: 15.967	MSE: 15.755	MSE: 15.485
7		MSE: 8.563	MSE: 8.73	MSE: 8.719
8		MSE: 4.612	MSE: 4.664	MSE: 4.619
9		MSE: 6.683 	MSE: 6.352 	MSE: 6.343 

Continued on next page

Test	Ground truth	Canny	CORF (1, 2, 1.8)	CORF (1, 4, 4)
10		MSE: 10.983	MSE: 11.759	MSE: 11.985
11		MSE: 6.245	MSE: 6.293	MSE: 6.278
12		MSE: 8.703	MSE: 9.283	MSE: 9.358
13		MSE: 15.495	MSE: 16.316	MSE: 15.852
14		MSE: 7.051	MSE: 7.328	MSE: 7.313
15		MSE: 14.365	MSE: 15.141	MSE: 15.112
16		MSE: 5.249 	MSE: 5.341 	MSE: 5.34 
17		MSE: 8.025	MSE: 8.319	MSE: 8.306
18		MSE: 12.918	MSE: 13.534	MSE: 13.441
19		MSE: 8.522	MSE: 8.592	MSE: 8.507

20		MSE: 17.473	MSE: 18.01	MSE: 17.96

Table 1: Regular test results (continued)

In Figure 13 we can see all MSE values of the regular test.

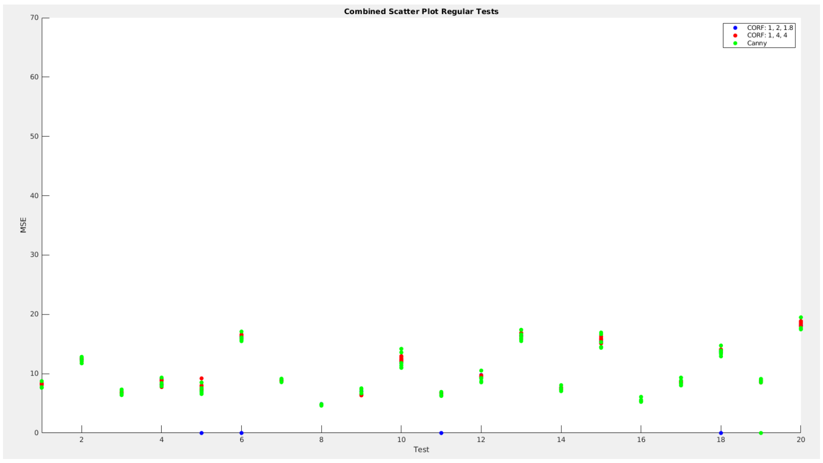


Figure 13: Regular test with all MSE values

In Figure 14 we can see the best MSE values for each test case.

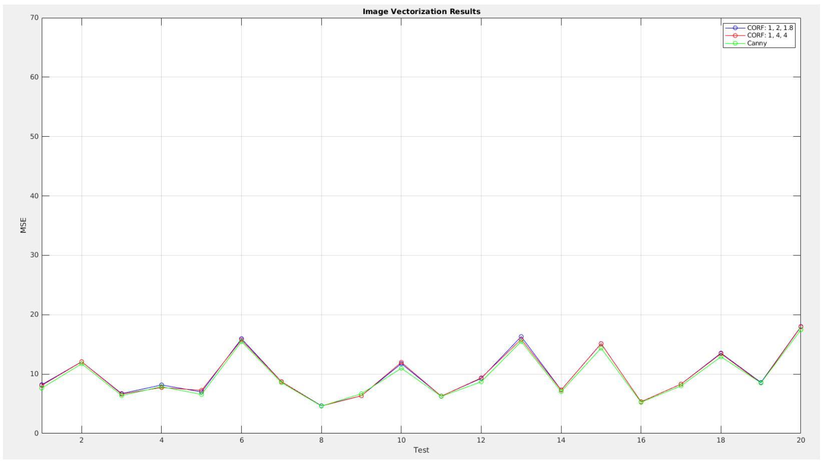
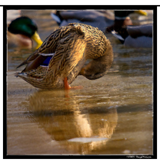
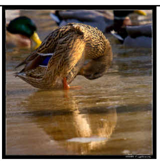
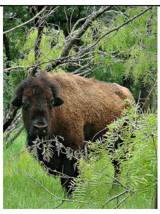
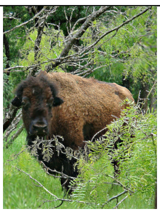
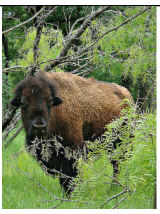


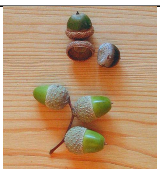
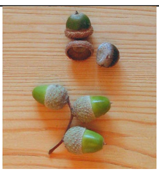
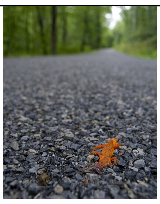
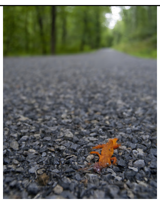
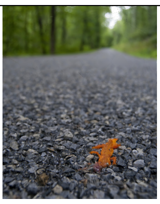
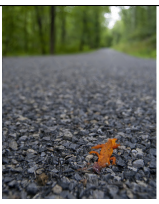
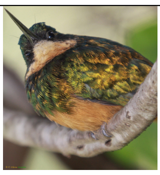
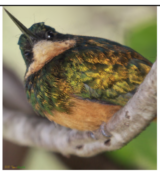
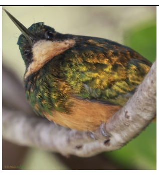
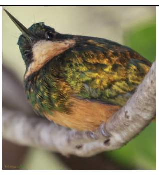
Figure 14: Regular test with best MSE values

5.3.2 Noise test

In Table 2 we show the 20 noisy image ground truths and the corresponding best outputs for each pipeline over the set of threshold values.

Noisy test images				
Test	**Ground truth**	**Canny**	**CORF (1, 2, 1.8)**	**CORF (1, 4, 4)**
1		MSE: 14.351	MSE: 14.411	MSE: 14.249
2		MSE: 15.049 	MSE: 14.967	MSE: 14.546 
3		MSE: 21.414	MSE: 21.424	MSE: 20.489
4		MSE: 22.323	MSE: 22.012	MSE: 21.197
5		MSE: 27.575	MSE: 29.658	MSE: 29.503
6		MSE: 8.351	MSE: 8.593	MSE: 8.524
7		MSE: 7.587	MSE: 7.746	MSE: 7.795
8		MSE: 10.029	MSE: 9.925	MSE: 9.927
9		MSE: 54.875 	MSE: 53.915	MSE: 52.739 

Continued on next page

Test	Ground truth	Canny	CORF (1, 2, 1.8)	CORF (1, 4, 4)
10		MSE: 24.393	MSE: 24.069	MSE: 23.504
11		MSE: 8.235	MSE: 8.217 	MSE: 8.786
12		MSE: 51.371	MSE: 50.093	MSE: 48.552
13		MSE: 10.588 	MSE: 10.549 	MSE: 10.09 
14		MSE: 10.41 	MSE: 9.86 	MSE: 9.727 
15		MSE: 10.241	MSE: 8.262	MSE: 8.269
16		MSE: 15.009	MSE: 14.463	MSE: 14.296
17		MSE: 10.649	MSE: 10.762	MSE: 10.68
18		MSE: 24.755	MSE: 24.899	MSE: 23.986
19		MSE: 22.188	MSE: 22.208	MSE: 21.539

Continued on next page

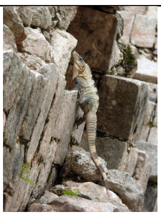
Test	Ground truth	Canny	CORF (1, 2, 1.8)	CORF (1, 4, 4)
20		MSE: 24.754	MSE: 24.69	MSE: 23.667 

Table 2: Noise test results (continued)

In Figure 15 we can see all MSE values of the noise test.

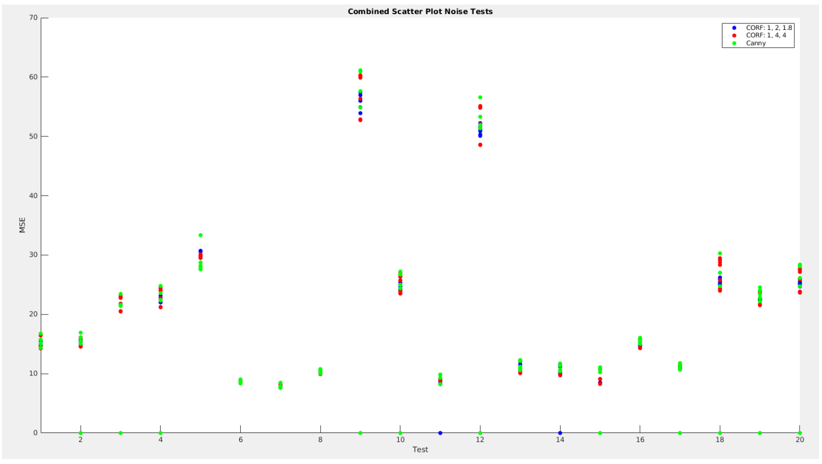


Figure 15: Noise test with all MSE values

In Figure 16 we can see the best MSE values for each test case.

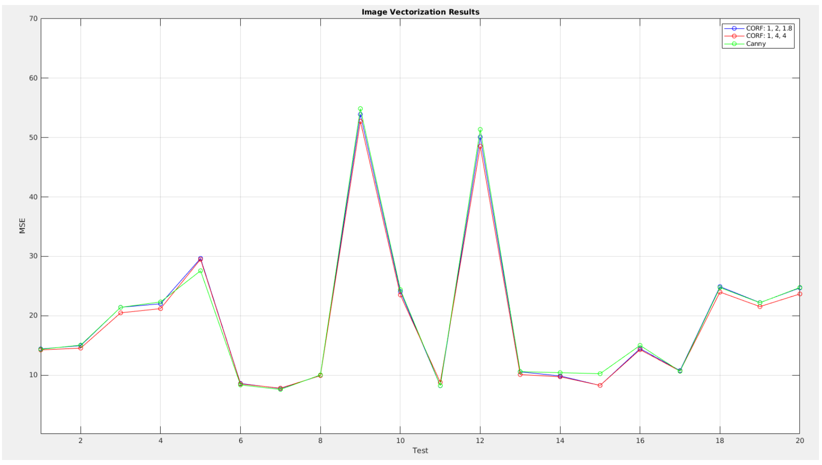


Figure 16: Noise test with best MSE values

5.3.3 Leopard test

Figure 17 shows the best results for each pipeline setup.

(a) Canny: 11.364 (b) CORF (1, 2, 1.8): 11.242 (c) CORF (1, 4, 4): 11.089

Figure 17: Leopard test MSE values

6 Discussion

Test	Canny	CORF (1, 2, 1.8)	CORF (1, 4, 4)
Noise	4	3	13
Regular	17	0	3

Table 3: Best performing pipelines per test

As we can observe in Table 3 when testing with regular images Canny produces on average a lower MSE compared to either of the two CORF versions. On the other hand, when testing with noisy images CORF - and more specifically CORF with the settings (1, 4, 4) - produces on average the lowest MSE.

Looking closer at the test cases where CORF outperforms Canny for regular images we can look at test case 4 and 9. We can look at the spread in Figure 13 to check if these test cases are too noisy to be in the regular test. We observe some spread between the data points, but not significantly more than the others in that test. Also, the MSE of the image is similar of those in the same test. Hence, we cannot say that these two images belong to the wrong test. During visual inspection of test case 4 we can see that there are a lot of undefined lines present, and so that is probably the root cause of why CORF outperforms Canny on this occasion. For test case 9 we can observe that the Canny-based result show various colorbleedings around the bounding box within the image, which is most likely the cause of why CORF outperforms Canny.

Taking a closer look at the test cases where Canny outperforms CORF for noisy images, we can look at test case 5 and 11. In Figure 15 we can make an interesting discovery. Although test case 5 shows significant spread and a decently high MSE, test case 11 has one of the lowest MSE of the test with a small spread. Hence, we can say that test case 11 is not as noisy as the rest of the test cases of the noise test, which is why Canny can produce a good result here.

Therefore, we can say that Canny produces on average a lower MSE on regular images and that CORF produces on average a lower MSE on noisy images. As we have only tested with small images (a maximum of 500x500) the running time of both of the algorithms did not make a significant impact. It is unsure how much more the undefined line and the colour bleeding problem is increased when using a CORF-based pipeline; what we can say is that on a simple circle, colour bleeding would occur with a CORF-based pipeline and not with a Canny-based pipeline. We have assumed that colour bleeding occurs more when using CORF. Although this is the case, CORF still outperforms Canny on noisy images, so resolving these issues will theoretically help CORF even more.

Another measure to look at is how consistent the pipelines perform. In other words, how often do the pipelines fail to produce a result for both the regular and the noisy images. If the pipeline fails to produce a result, it is caused by either too many discontinuous edges or too many edges in general. This causes the triangulation step to fail while inserting new vertices to create new triangles, as there are too many triangles that have to be created. In both Figure 15 and 13 we can observe MSE values of 0; this means that the pipeline did not produce a result for that specific setting. In total CORF with the settings (1, 2, 1.8) failed to produce a result 21 out of 320 times, CORF with the settings (1, 4, 4) failed to produce a result 2 out of 320 times and Canny failed to produce a result 16 out of 240 times. The best fail ratio is achieved by CORF with the settings (1, 4, 4) with a ratio of 0.63%, followed by CORF with the settings (1, 2, 1.8) with a ratio of 6.56% and Canny with a ratio of 6.67%. Hence, CORF with the settings (1, 4, 4) can consistently produce at least a vectorized result without failing the triangulation step, which means it is the best out of the 3 for reliability.

We have previously concluded that the contour map of Canny and CORF differ significantly, of which the comparison can be seen in Figure 1. The best vectorized results for each pipeline can be found in Figure 17. As can be seen, the MSE values for both variants of the CORF-based pipeline are significantly lower. Hence, also for this specific case a CORF-based pipeline can outperform a Canny-based pipeline.

As for future work, a measure can be created to combine the MSE and the compression ratio to create a more general view of the produced results. In reaching a conclusion the actual usability of the produced results is currently not taken into account. Hence, the file size of the produced results can be too large to be used, or the mesh can be too complex to work with. Additionally, solving both the undefined line and the colour bleeding problem will increase the accuracy of the MSE. One can say that we are now choosing the method where these problems occur the least, not necessarily producing the best result. Furthermore, we have used the same edge detector settings for both the input image and the quantized grayscale version of the image, which was not the case for the default pipeline settings. Therefore, adjusting these to find a balance between the two can yield a significant increase in performance.

7 Conclusion

To go back to the hypothesis made in the introduction we can look at Figure 17. As previously hypothesized a CORF-based pipeline might be able to outperform a Canny-based pipeline when presented with a heavily textured input image. To reiterate, the edge detector push/pull CORF would produce less edges on heavily textured images compared to Canny's edge detector. During our research we found that the more textured an image is, the more important it is to not place edges around all the textures' intricacies. Take for example the leopard mentioned in the introduction and analyzed in the discussion; the quality of the produced result is better when using an edge map that is closer to the ground truth. Hence, when trying to vectorize a heavily textured object, having only the outline of the object as an edge is more important than an edge that captures all the details. Additionally, fewer edges mean a less complex mesh, which results in a smaller file size and a vector image that is easier to work with. Furthermore, fewer edges also mean that the mesh is easier to create, which results in more reliable triangulation and less computing power.

For regular imagery, a Canny-based pipeline still outperforms a CORF-based pipeline. As Canny is really good at placing edges at places where colour differences occur, the resulting edge map is very close to the ground truth. Hence, in the absence of heavy textures, the derived vectorized result is the best.

Therefore, we can conclude that a CORF-based image vectorization pipeline can outperform a Canny-based image vectorization pipeline when presented with significantly noisy images, such as white Gaussian or natural noise in the form of heavy textures. This results in a smaller difference to the ground truth, a more reliable pipeline, and a less complicated mesh.

Based on the research done in this thesis, we can identify a set of future improvements. For one, the MSE and the compression ratio value can be combined into a single measure. This results in a more accurate interpretation of the 'best' vectorized result, as only using the MSE can give a limited view and does not capture the whole picture. Additionally, both the colour bleeding and the undefined line problem could be solved, to decrease the current inaccuracies and to bring down the MSE value. Furthermore, this will make the comparison more accurate. As a result of this research, we can now supply any kind of contour map to the image vectorization pipeline. Hence, this new structure can also be used to feed in artificially made contour maps to further understand why these colour bleedings and undefined lines occur. Another improvement could be to use a different edge detector setting for both the input image and the quantized grayscale version of the image. To contain the amount of edge detector settings during this research, we have set these to the same setting. Finally, a new edge detector could be integrated into the pipeline to create one entity. This would allow for easier testing of different settings and would allow for different ways to compare the vectorized results.

Bibliography

[1] Y. Liu, M. M. Cheng, X. Hu, K. Wang, and X. Bai, “Richer Convolutional Features for Edge Detection.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, vol. 41, pp. 1939–1946, 2019.

[2] G. Azzopardi, A. Rodríguez-Sánchez, J. Piater, and N. Petkov, “A Push-Pull CORF Model of a Simple Cell with Antiphase Inhibition Improves SNR and Contour Detection.,” *PLOS ONE*, vol. 9, p. np., 2014.

[3] D. Ziou and S. A. Tabbone, “Edge Detection Techniques - An Overview.,” *Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications.*, vol. 8, p. np., 2000.

[4] J. Canny, “A Computational Approach to Edge Detection.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, vol. 8, pp. 679–698, 1986.

[5] G. J. Hettinga, J. Echevarria, and J. Kosinka, “Adaptive image vectorisation and brushing using mesh colours.,” *Computers Graphics.*, vol. 105, pp. 119–130, 2022.

[6] M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun, “Effective Clipart Image Vectorization through Direct Optimization of Bezigons.,” *IEEE Transactions on Visualization and Computer Graphics.*, vol. 22, pp. 1063–1075, 2016.

[7] Y. B. Bai and X. W. Xu, “Object Boundary Encoding — a new vectorisation algorithm for engineering drawings.,” *Computers in Industry.*, vol. 46, no. 1, pp. 65–74, 2001.

[8] X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov, Y. Fu, and H. Shi, “Towards Layer-Wise Image Vectorization.,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*, pp. 16314–16323, June 2022.

[9] J. J. Zou and H. Yan, “Cartoon image vectorization based on shape subdivision.,” *Proceedings. Computer Graphics International.*, pp. 225–231, 2001.

[10] A. Bera, “Fast vectorization and upscaling images with natural objects using canny edge detection.,” *3rd International Conference on Electronics Computer Technology.*, vol. 3, pp. 164–167, 2011.

[11] G. A. Orban, “Higher order visual processing in macaque extrastriate cortex.,” *Physiological reviews.*, vol. 88, pp. 59–89, 2008.

[12] K. P. Hoffmann and B. Dreher, “Properties of excitatory and inhibitory regions in the receptive fields of single units in the cat’s superior colliculus,” *Exp Brain Res*, vol. 16, pp. 333–353, 1973.

[13] G. Azzopardi and N. Petkof, “A CORF computational model of a simple cell that relies on LGN input outperforms the Gabor function model.,” *Biol Cybern*, vol. 106, pp. 177–189, 2012.

[14] G. Azzopardi, *Contour Detection with the Push Pull CORF Model.* MATLAB Central File Exchange., 2023.

[15] A. Howard, E. Park, and W. Kan, *ImageNet Object Localization Challenge.* Kaggle., 2018.