Data Science Project

Automated Lineament Detection from Digital Elevation Models (DEM) & Field Pictures

Le-Bourget-du-Lac (France), 16.08.2022

Abstract

In geological sciences, it is of major interest how faults and folds are oriented in nature, since they can host potentially hazardous earthquakes. In this design report, we discuss how various types of input images (drone photos, Lidar-images, orthophotos) can be used to obtain lineaments (e.g. faults) in an automated way with a python-based algorithm and compare these results to a reference dataset created by a geologist. The different outputs (lineaments) are then compared to each other using performance indexes (such as the Jaccard score for image similarity) and are statistically analyzed. The output is additionally clustered with various methods such as k-means and dbscan. We investigate potential gains of having an automated algorithm and try to highlight the benefits of introducing more automatism and subjectivity to the geologist's job.

Stefano Fabbri
Liebefeld
Switzerland
stefano.fabbri@geo.unibe.ch
Wabersackerstrasse 23
3097

Table of Contents

1 Project Objectives

In various disciplines of geological sciences, e.g. tectonics, structural geology, geomorphology, paleoseismology, petrology, it is of major interest how geological faults and folds are oriented in nature. Lithospheric plate motion inherently leads to collision of continents, stresses are applied on material, strain builds up until material strength is exceeded and the accumulated strain energy is suddenly released in the form of mechanical failure, commonly known as earthquakes (Burbank and Anderson, 2011). Switzerland is prone to potentially strong earthquakes, as it lies at the heart of the collision zone between the Adriatic and the European plate (Wiemer et al., 2009). These mechanical failures, where earthquakes occurred in the past and possibly occur again in the future, are also known as faults, recognizable as straight lines on digital elevation models and field photographs. The extraction of straight linear topographic features (lineaments) from satellite images and drone pictures, e.g. Digital Elevation Models (DEM) derived from Light Detection and Ranging (LiDAR) images, and from field photographs (FIPH) at outcrop scales and orthophotos, is a common instrument to identify overall, ideally fault-based, trends of tectonic elements (Smith and Pain, 2009). Orthophotographs are aerial photograph or satellite imagery geometrically corrected such that the scale is uniform: the photo or image follows a given map projection; taken form: *https://en.wikipedia.org/wiki/Orthophoto.*

Usually, geologists pick these lineaments by hand, which is a very tedious work with high subjectivity. Automatically extracted lineaments are a valuable contribution to the tectonic investigation of a mountainous area, especially since subjectivity is drastically reduced, prior training of geologists not required and efficiency and reproducibility improved. The author wrote as part of his dissertation a *MATLAB*-based algorithm which automated the lineament identification in DEMs and FIPHs. In the framework of this thesis, a python routine has been developed to successfully retrieve lineaments from a digital elevation model and orthophotos. The dataset from the python routine is then compared with the datasets created by the *MATLAB*-based algorithm. Furthermore, a geologist manually interpreted the orthophotos and created a dataset of lineaments, which will serve as a reference dataset. The produced dataset of lineaments are used as input for further analysis and statistical evaluation, including some unsupervised machine-learning algorithms for further interpretation of data, which is the main purpose of this project.

Hence, the aim of this CAS project is multi-fold, is split into 4 goals that want to

a) Plot the input data and test various image enhancement techniques applied to the input data (satellite and drone based imagery)

b) Create a python routine to retrieve lineaments (output) from various satellite and drone based imagery (input) using edge detection algorithms

c) Analyze the output (lineaments) from DEMs and FIPHs

d) Use unsupervised machine-learning algorithms to interpret data and compare the results with data from hand-picked (geologist) datasets

Each goal has its own dedicated python-script that is stored on github and named after the goal (e.g. Goal_C_DataAnalysis.ipynb).

In **a)**, we will have to perform some pre-processing steps individually, shape the input data, plot the input data in various fashions and test image enhancement techniques (e.g. filtering using convolutional models and fast-fourier transform, contrast enhancements, different degrees of image blurring). Image blurring usually helps to perform better edge detection results. For goal **b)**, an edge-detection algorithm was created to retrieve linear features from the pre-processed input data. The output will be straight lines with x,y- coordinates. In **c)**, we remove invalid values and duplicates from the output data and carry out some further analysis, including some descriptive statistics (counts, orientation, mean, standard deviation, median etc.) using histograms and rose diagrams, and directional statistics (often used when dealing with circular or spherical information as in this case). Goal **d)** deals with the interpretation of the output using unsupervised machine-learning techniques, such as k-means clustering, applied to 4 different datasets. i) the reference dataset manually hand-picked by a geologist, ii) the *MATLAB*-based output, iii) the edge-detection output from python, based on an orthophoto as input and iv) the edge-detection output from python, based on an multi-directional hillshade image as input.

Potential shortcomings (e.g. misinterpretation of man-made infrastructure by the algorithms) have been removed, since the study area is simply 20-by-20 m large and does not show any man-made or artificial constructions. The man-made reference dataset was mostly created using the orthophoto as primary source for interpretation and only to a very minor extent has this interpretation be cross-checked with DEM-derived hillshade images. Illumination angle and additional algorithm-related input parameters (filtering, type of edge detection method, amount of peaks retrieved in the Hough domain, etc.) need to be chosen adequately and will be discussed in details in Goal a) and b).

The reader should be aware that the primary results of this thesis are the python-based algorithms and data analysis codes and scripts. Hence, this written short thesis summarizes the most important findings and figures, and more details can be taken from the 4 python codes related to the 4 goals of this thesis. Important coding pieces and snippets are added to this thesis where relevant and are marked with black background to separate it from the rest of the manuscript.

2 Methods

Infrastructure

The coding was done with ***Google Colab,*** hence it was running on cloud services and was not dependent on a local machine. Only a internet access and a basic computer is required for that purpose. Most of this work was performed at the Institute of Geological Sciences, University of Bern, Switzerland. Notebooks and input data were stored on ***Google Drive*** and on ***GitHub.***

"Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate.

In order to be able to offer computational resources free of charge, Colab needs to maintain the flexibility to adjust usage limits and hardware availability on the fly. Resources available in Colab vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors." (from https://research.google.com/colaboratory/faq.html)

Software

The coding was done to minor extent in ***MATLAB® 2013a*** and to a large amount in ***Google Colab*** (hereafter referenced as python). For all goals, python was used for plotting, illustrating, data analysis and *MATLAB® 2013a* was only used for one dataset creation in order to compare it with dataset created with python (comparison in Goal c) and d). *MATLAB®* was used on a local machine and requires a license. Some minimum performance requirements have to be met. 4 GB of RAM is required, but 8 GB with 4 GB per core is recommended. Minimum Processor required: Any Intel or AMD x86-64 processor.

ArcMap 10.8.2 from the ArcGIS suite was used for the creation of the input data in the form of i) orthophotos from drones, ii) digital elevation models from LiDAR or drone imagery, and iii) DEM-derived products thereof such as hillshades. Furthermore, ArcMap was used to define the perimeter of interest and ensure the datasets are geo-referenced.

The author wrote an automated lineament detection algorithm in *MATLAB® 2013a* (hereafter termed "matlab" for simplicity) and used mostly existing internal functions. The flowchart of the detection and mapping algorithm for structural lineaments (Fig. 1) follows largely the recommendation of Rahnama and Gloaguen (2014b, 2014a), since they have tested various combinations of filters, edge detection algorithms and parameter setups for the Standard Hough Transform (SHT). The tensor voting algorithm was developed by Linton (2008). The matlab-based algorithm outputs spatial

coordinates of every start and end point of each lineament (see also chapter 3). This information will be used for further analysis and comparison with the python-based algorithm and the reference dataset.

Libraries & statistical methods

The list of libraries are declared at the beginning of each python script, therefore, please refer to the scripts for details. Also all applied statistical methods are defined in the code scripts.

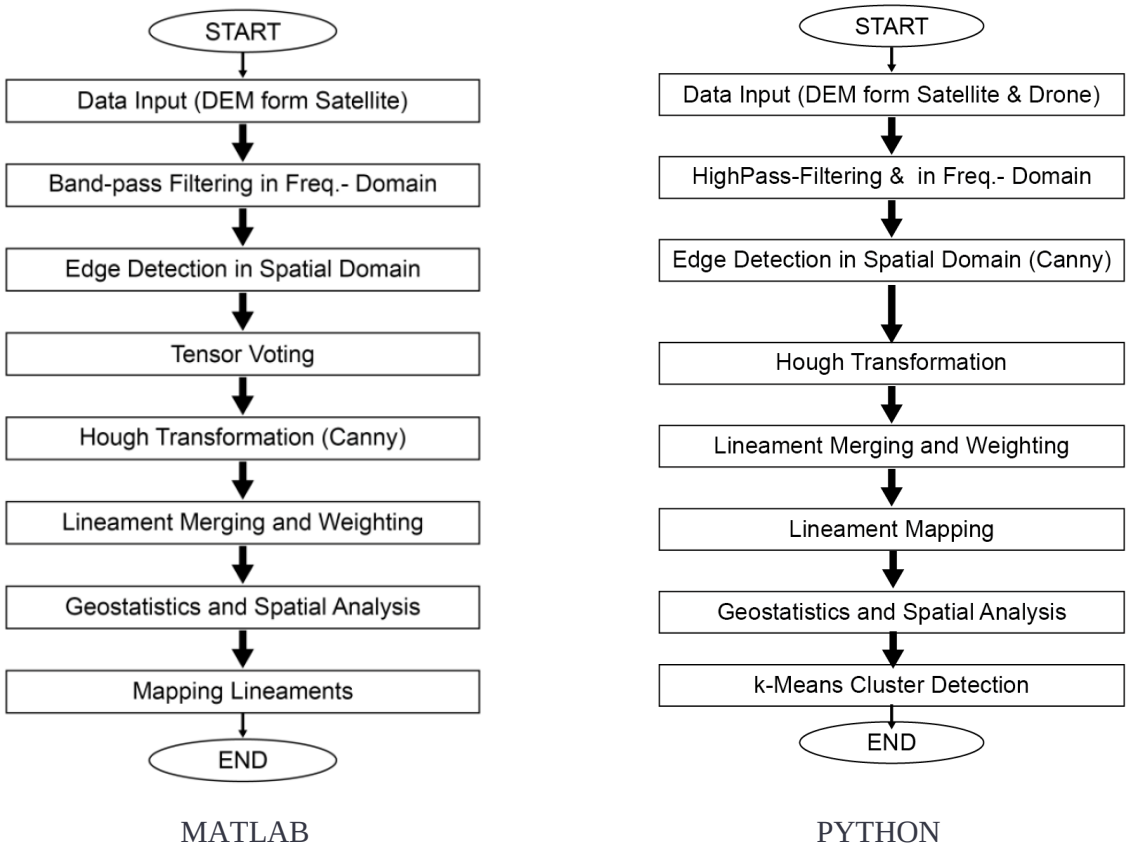

Fig. 1, left: Overview of processing flow for automated lineament detection using matlab. Right: Overview of processing flow for automated lineament detection using Google Colab. Note that there is no tensor voting applied in the python-based workflow.

3 Data

The data that serves as input for analysis and algorithm-testing has been collected by Sandro Trutmann (PhD student at the Institute of Geological Sciences, University of Bern). The original datasets are confidential and all rights belong to the University of Bern. The geological area of interest lies in the canton of Valais (Switzerland).

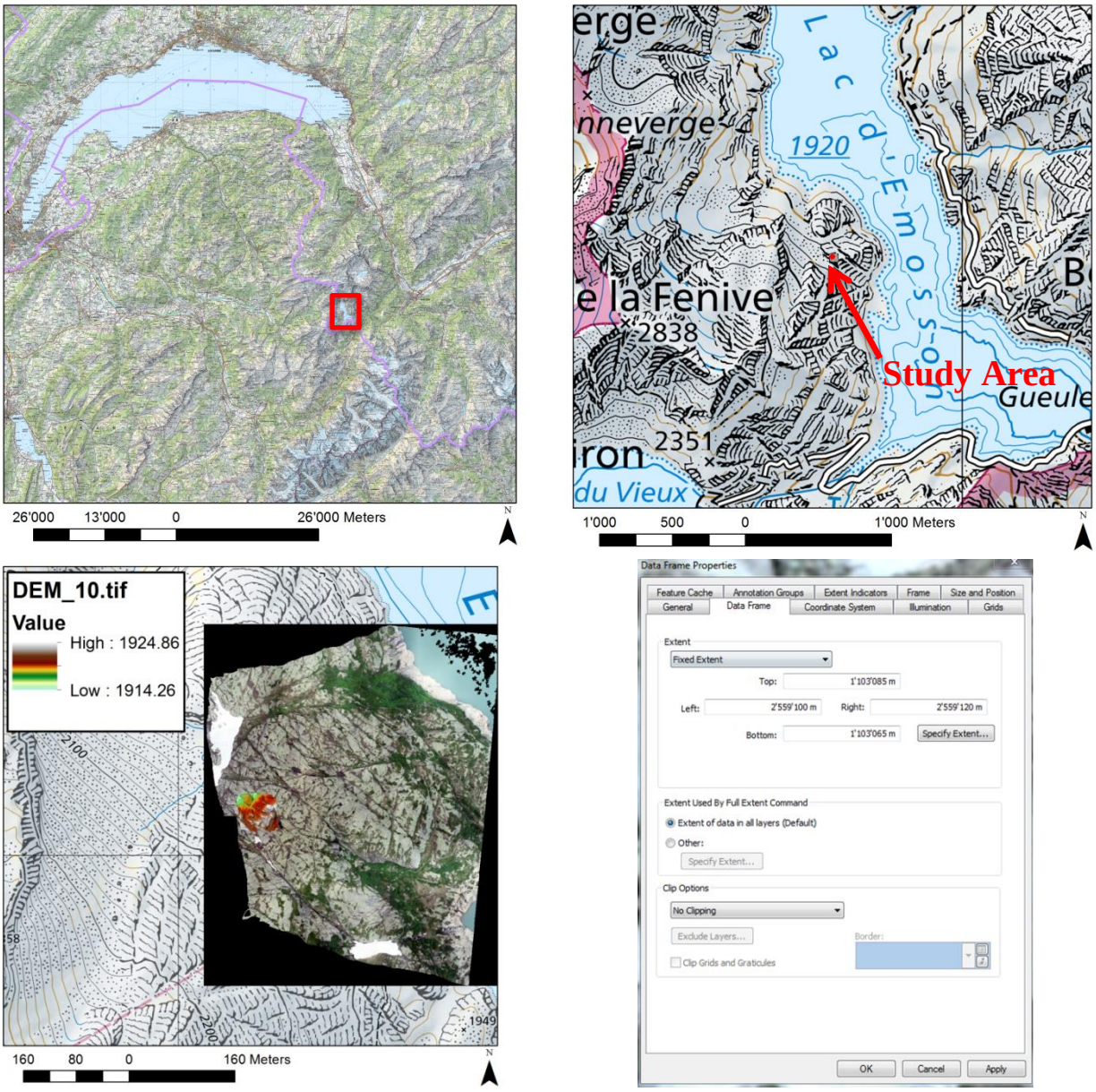

Fig. 2, top left: Overview of Southwestern Switzerland. Top right: Zoomed version of left, with "Study Area" indicated. Bottom: 2 screenshots from ArcMap. The study area is covered by the color-coded digital elevation model (physical coordinates of the perimeter of interest: top left corner: 2'559'100/ 1'103'085; bottom right corner: 2'559'120/ 1'103'065, CH1903+/LV95 coordinates) and shows the investigated area used for data creation. The orthophoto from drone shots is shown in the background. It covers an area of 20-by-20 meters and was exported with a resolution of 2400 by 2400 pixels.

Figure 2 gives an overview of the area of interest that covers roughly 20-by-20 meters and lies around 1920 m a.s.l. The exact area that is fed to the algorithm can be taken from the coordinates in Fig. 2 (top left and bottom right edges of extracted area). The coordinates and additional information about the selected area is given in the caption of figure 2.

3.1 Goal A – Data Loading and Image Enhancement

The data of the perimeter was loaded into python as csv-files (DEM) or as tif-files (Hillshades & Orthophotos). Figure 3 displays the input to python for further data analysis. The files were stored on *Google Drive* and loaded from there in to python.

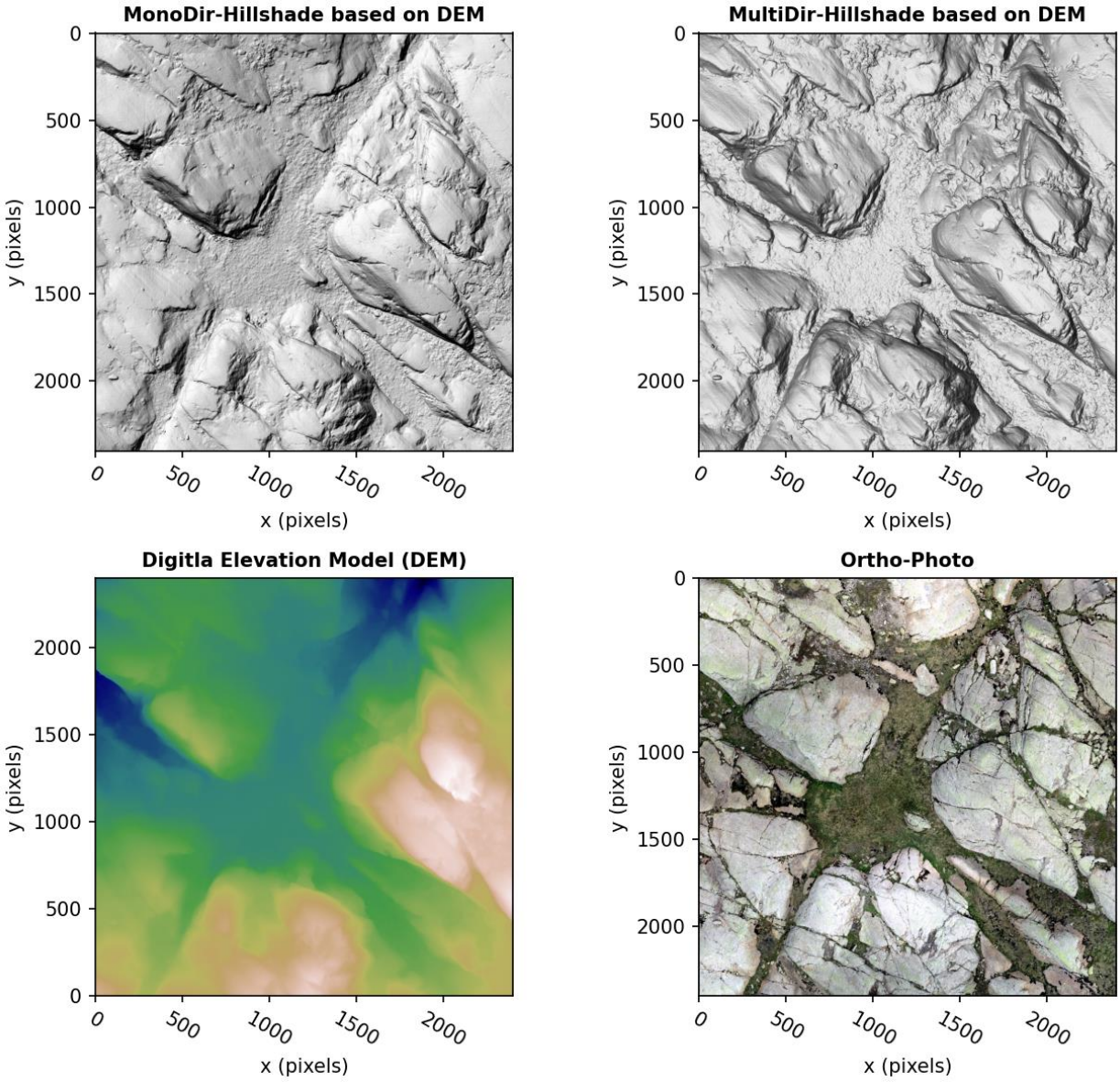

Fig. 3: Dataset and image loading (Goal A).Note that spatial coordinates are pixels and not physical coordinates.

The DEM information and the Hillshade can also be combined, as shown in Figure 4. The Hillshade image is used as background and the color-coded elevation information is superimposed. With the aim to detect lineaments and edges in our input data, the orthphoto shown in Fig. 3 was converted to gray-scale and compared to a contrast-enhanced image of the orthophoto (Fig. 5). Enhanced contrast will likely perform better when using an edge detection algorithm, since its detection is based on the gradient magnitude image from one pixel to its neighboring pixel.

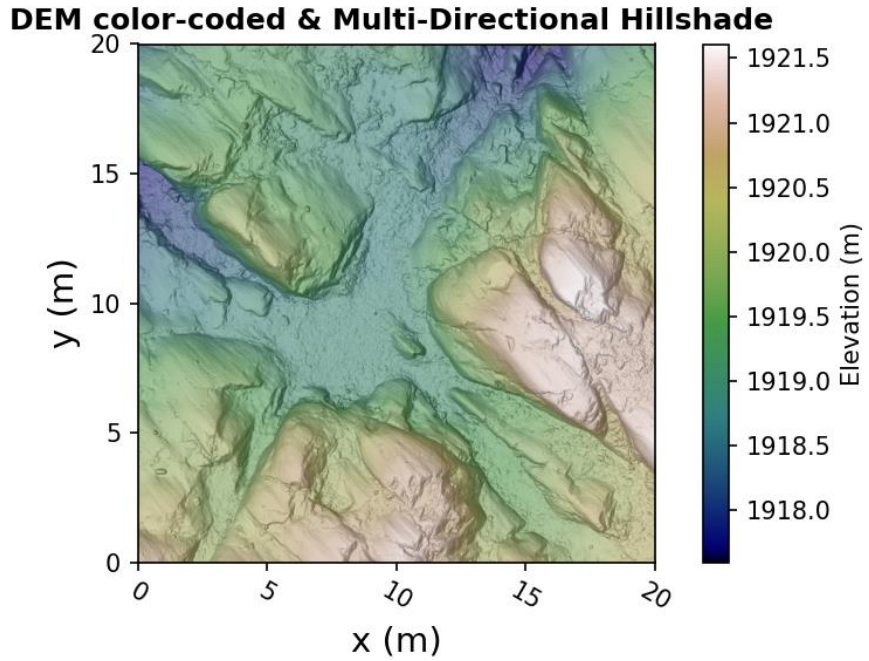

Fig. 4: Data loading, combining elevation information from a digital elevation model and a DEM-derived multi-directional Hillshade. Note, the pixel has been converted to physical distance in meters.

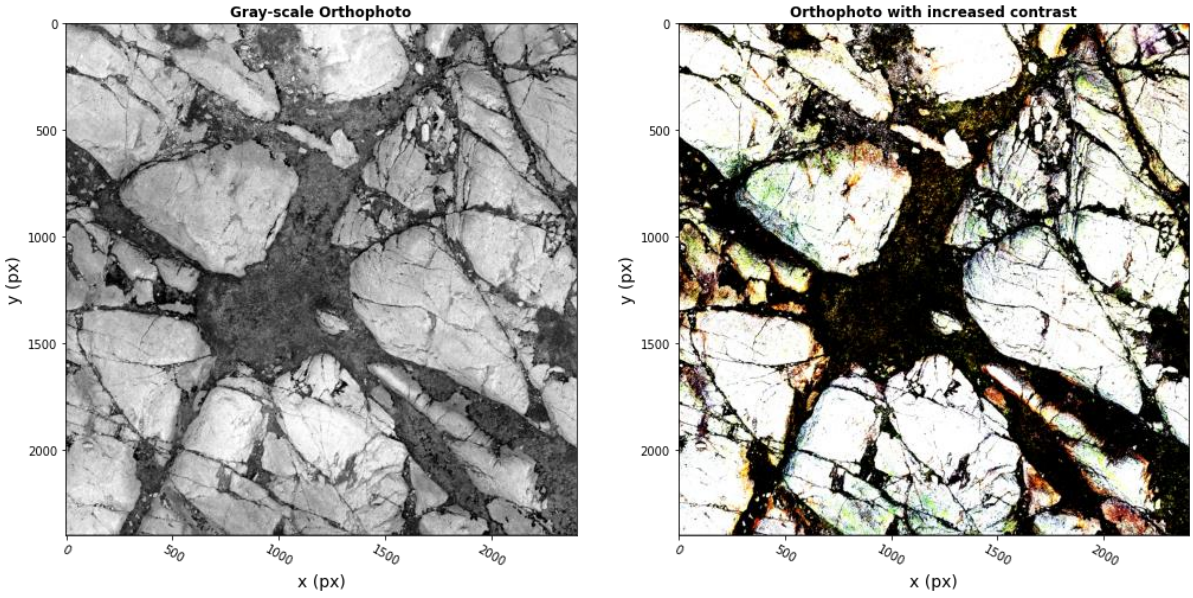

Fig. 5, Left: Orthophoto changed to gray-scale; Right: Orthophoto with increased contrast to amplify linear feature in the form of faults.

Best results are achieved by combinin the contrast-enhanced orthophoto the gray-scale image, as shown in Figure 6. The orthophoto was contrast enhanced, then converted to gray-scale and then inverted. This amplifies the linear features in the image, which are of primary concern and interest in this thesis.

```
# This is an inverse operation of the grayscale image, you could see that
the

# bright pixels become dark, and the dark pixels become bright

# Gray Image

gray_image_contrast = cv.cvtColor(ortho_contrast, cv.COLOR_BGR2GRAY)



# Inverse Gray Image

inv_gray_contrast = 255 - gray_image_contrast
```

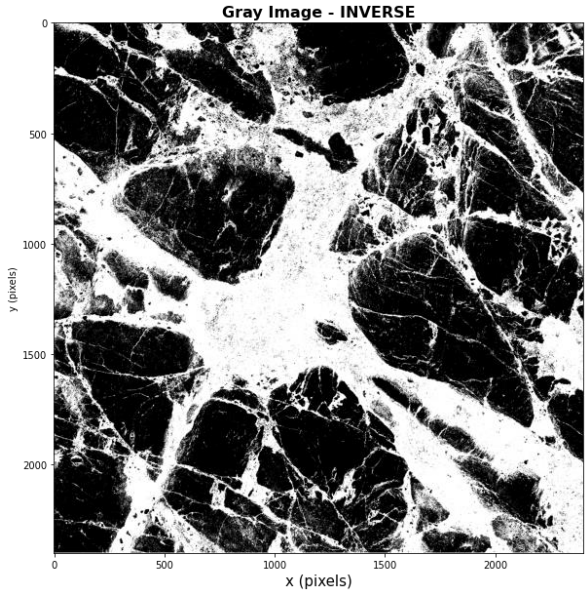

Fig. 6: Contrast enhanced image, with inverted gray-scale scheme.

As a next step, we perform a fourier transform, which is used to find the frequency domain of an image. You can consider an image as a signal which is sampled in two directions. So taking a fourier transform in both X and Y directions gives you the frequency representation of the image. For the sinusoidal signal, if the amplitude varies so fast in short time, you can say it is a high frequency signal. If it varies slowly, it is a low frequency signal. Edges and noises are high frequency contents in an image because they change drastically in images.

```
# Blur the inverse-
grayscale image by a Guassian filter with kernel size of 10

inv_gray_contrast_imBlur5 = cv.blur(inv_gray_contrast,(5,5))
```

```
# Transform the image to frequency domain

f = np.fft.fft2(inv_gray_contrast_imBlur5)

# Bring the zero-frequency component to the center

fshift = np.fft.fftshift(f)

magnitude_spectrum = 30*np.log(np.abs(fshift))
```

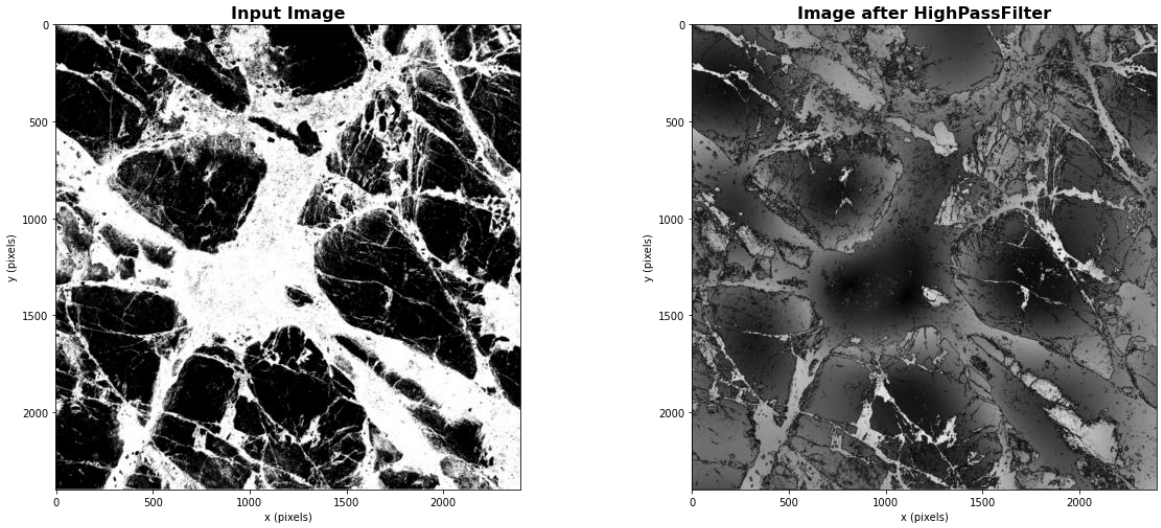

Fig. 7: Output of fast-fourier transform, eliminating low frequencies and let high frequencies pass, for better edge-detection performance.

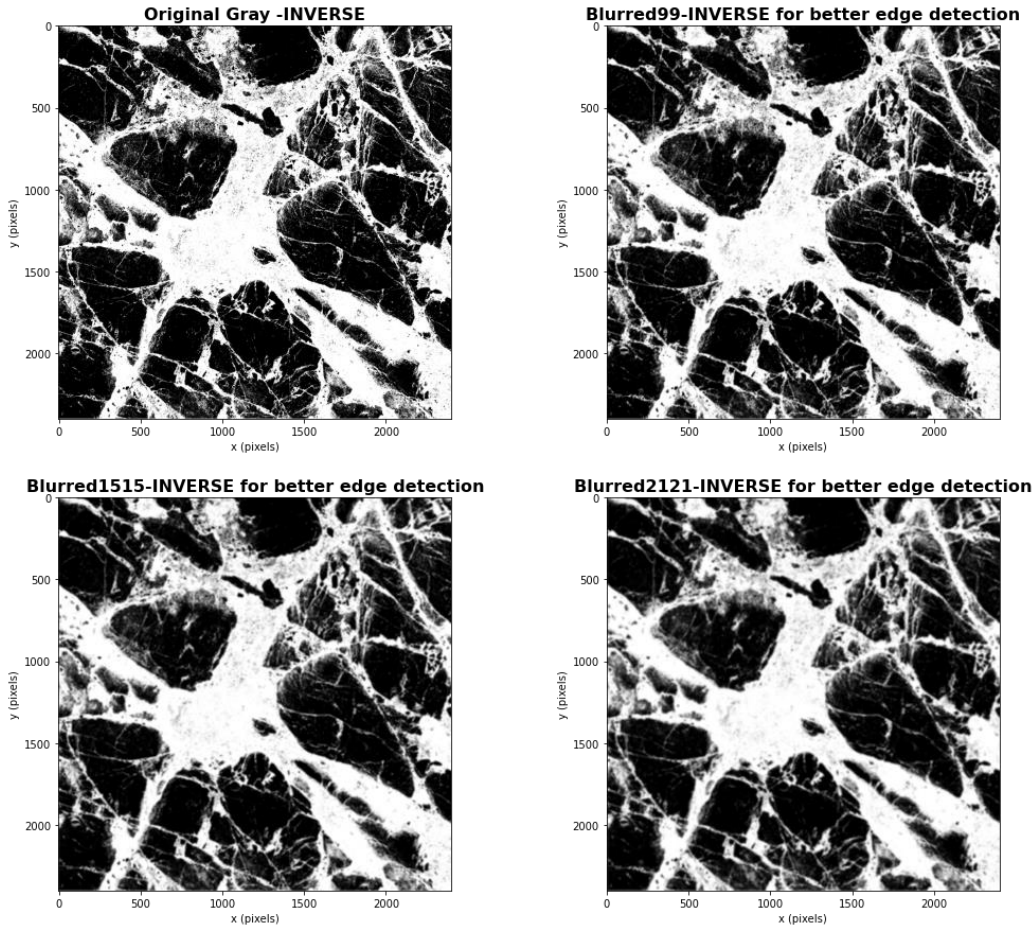

Fig. 8: Blurring the image using a Gaussian Kernel filter using 9-by-9 kernel, 15-by-15 kernel and 21-by-21 kernel.

The elimination of low frequency does not contribute significantly to better edge detection. Therefore, this step, applying fast-fourier transform, has been removed from the workflow, and images are only:

- contrast-enhanced
- converted to grayscale
- Inverted
- Blurred with a Gaussian Kernel Filter

We created various versions of blurring in order to test the best performance for edge detection. Additionally, a set of images has been produced, that are only constrast-enhanced, and blurred. Furthermore, a set of images was created that uses the Hillshade with various degrees of blurring.

3.2 Goal B – Edge Detection

First pre-processing steps shaped the input data, plot the input data in various fashions and tested image enhancement techniques (e.g. filtering using fast-fourier transform, contrast enhancements, different degrees of image blurring). Image blurring usually helps to perform best when applying edge detection algorithms.

A first attempt of edge detection was made with a convolution model that convolves a filter matrix with an image. Fig. 9 shows the model summary of this convolution with four convolutional layers. The TensorFlow library of Google with Keras API is used for this task. The filter matrix has been applied with different shapes and the results of this experiments are shown in more detail in python script “ Goal_B”). In figure 10, the output of the convolution of the pre-processed orthophoto with a diagonal filter matrix along 45 and 225 degrees is shown.

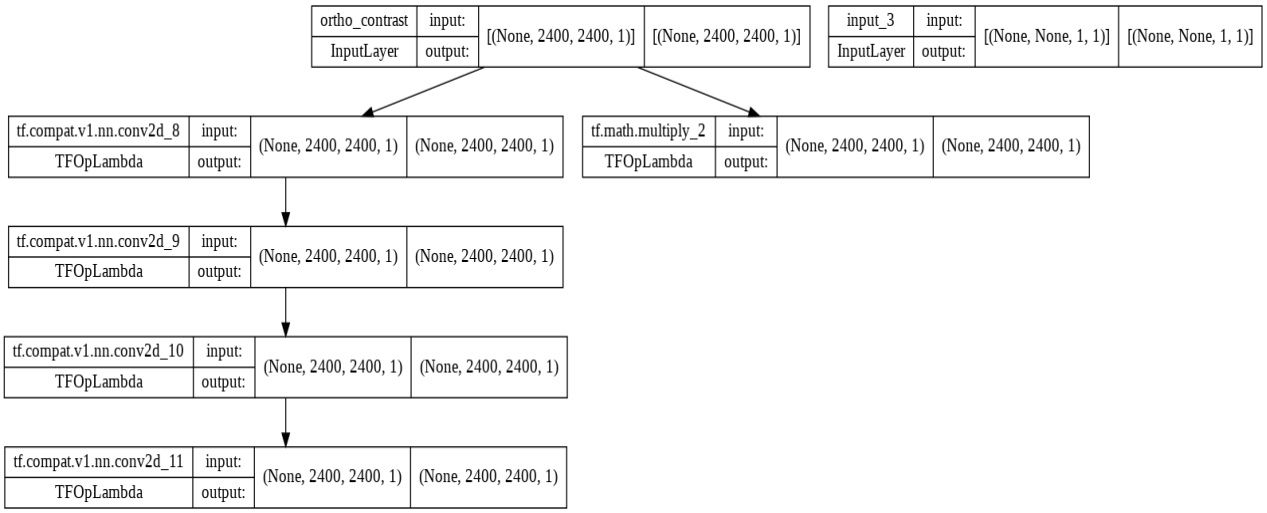

Fig. 9: Model summary of CNN-model and filtering process.

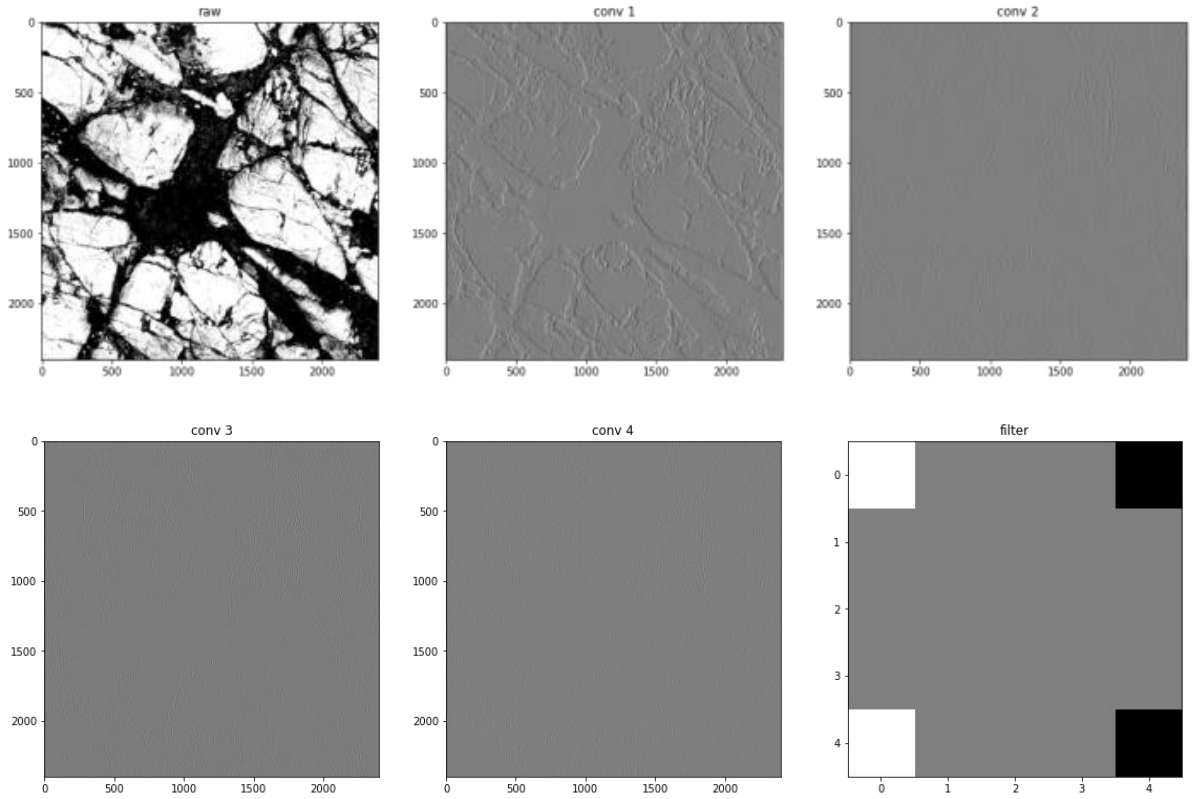

Fig. 10, top left panel: Input data (contrast enhanced orthophoto with a conversion to inversed gray scale. Conv 1 to 4 show the individual output of the convolution 1 to 4 times convolving with the photo. The panel "filter" indicates the shape of the applied filter matrix.

As second method and alternative to the convolutional model, Sobel edge detection can be implemented. It can be coded in different ways. Here we used the cv2 library to write the code. The detection can be performed in separate directions, e.g. only in x or y-direction, or in both directions combined (see Fig. 11).

"The Sobel Operator is a discrete differentiation operator. It computes an approximation of the gradient of an image intensity function. The Sobel Operator combines Gaussian smoothing and differentiation." (From https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html)

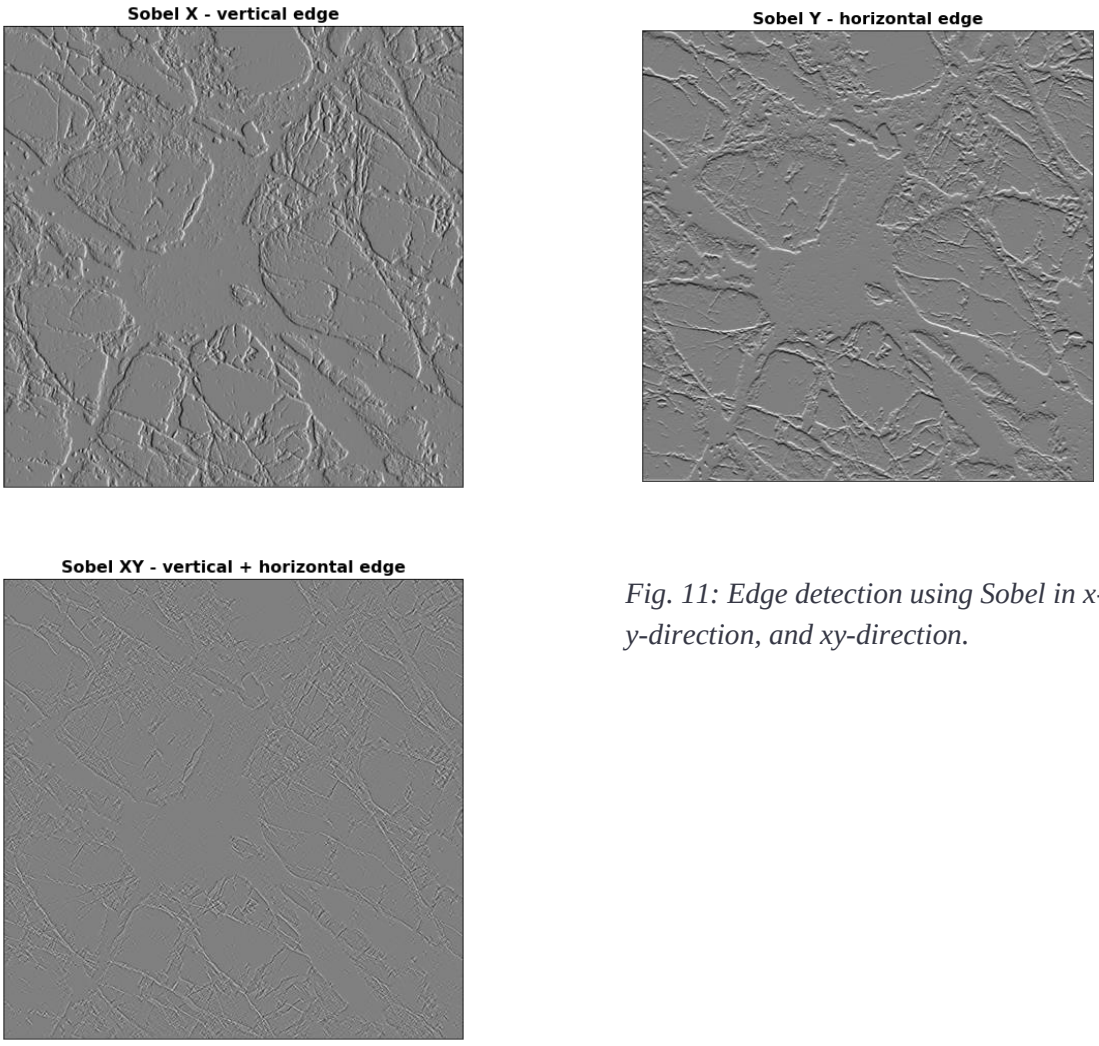

Fig. 11: Edge detection using Sobel in x-direction, y-direction, and xy-direction.

As a third method, we introduced the "Canny" edge detection algorithm, using cv library again. Testing of the different edge detection methods showed that "Canny performs best for type of input we are interested in. The heart of the code covers just one line of coding, and needs an image (format: unit8) as input, plus 2 different thresholds and 2 additional parameters (aperture size, L2 gradient).

```
edges = cv.Canny(image=image, threshold1=threshold1, threshold2=threshold2, apertureSize=3, L2gradient=True ) # Canny Edge Detection
```

The two threshold values, minVal= **threshold 1** and maxVal= **threshold 2** define if an edge with a certain intensity gradient is classified as true edge or not. *"Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are*

connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded." *(from https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html)*

We generally noticed that a threshold above 130 leads to no edge detections for our input.

L2gradient specifies the equation for finding the gradient magnitude. If it is set to "True", it uses the equation mentioned in (1). The input image is then filtered with a Sobel kernel in both horizontal and vertical direction to get the first derivative in horizontal direction (Gx) and in vertical direction (Gy). From these two images, we can find the edge gradient and direction for each pixel as follows:

$$Edge\ gradient\ G = \sqrt{G_x^2 + G_y^2} \quad (1)$$

$$Angle\ (\theta) = \tan^{-1}\frac{G_y}{G_x} \quad (2)$$

(modified from: https://docs.opencv.org/3.4/d7/de1/tutorial_js_canny.html)

We used L2 gradient and set it to "True", as it delivers better performance with Canny and removes many very short lineaments that are not of prior interest.

The output of the "Canny" edge detection algorithm is displayed in Figs. 12 and 13 below, using once the pre-processed orthophoto as input and once the hillshade as input, respectively. It is important to note, that the detected edges are at this stage not recognized as lines, they are simply stored as pixels. Hence, the output of the detection algorithm is a binary image.

Therefore, the pixel image has to be converted to line features (lineaments). This is done by the Hough Transform. Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. More theoretical details can be taken from: *https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html*

The relevant code is displayed below and requires the detected edges as input.

```
linesP = cv.HoughLinesP(edges,rho=0.5,theta=0.5*np.pi/180,threshold=10,minLineLength=20,maxLine
Gap=5) # Max line Gap very sensitive, used to reduce amount of lineaments and merge them

for x1,y1,x2,y2 in linesP[0]:

    cv.line(cedgesP,(x1,y1),(x2,y2),(0,255,0),2)
```

The OpenCV implementation is based on Robust Detection of Lines using the Progressive Probabilistic Hough Transform by Matas, J. and Galambos, C. and Kittler, J.V. (from: *https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html*)

The straight line in the image space is defined by the following parametric representation:

$$\rho = x * \cos(\theta) + y * \sin(\theta) \quad (3)$$

with ρ being the orthogonal distance of a vector from the origin to the line and θ the angle between the x-axis and this vector. Apart from the input (image) and ρ and θ, the function requires 2 additional input arguments for the probabilistic Hough Transform:

minLineLength - Minimum length of line. Line segments shorter than this are rejected.
maxLineGap - Maximum allowed gap between line segments to treat them as single line.

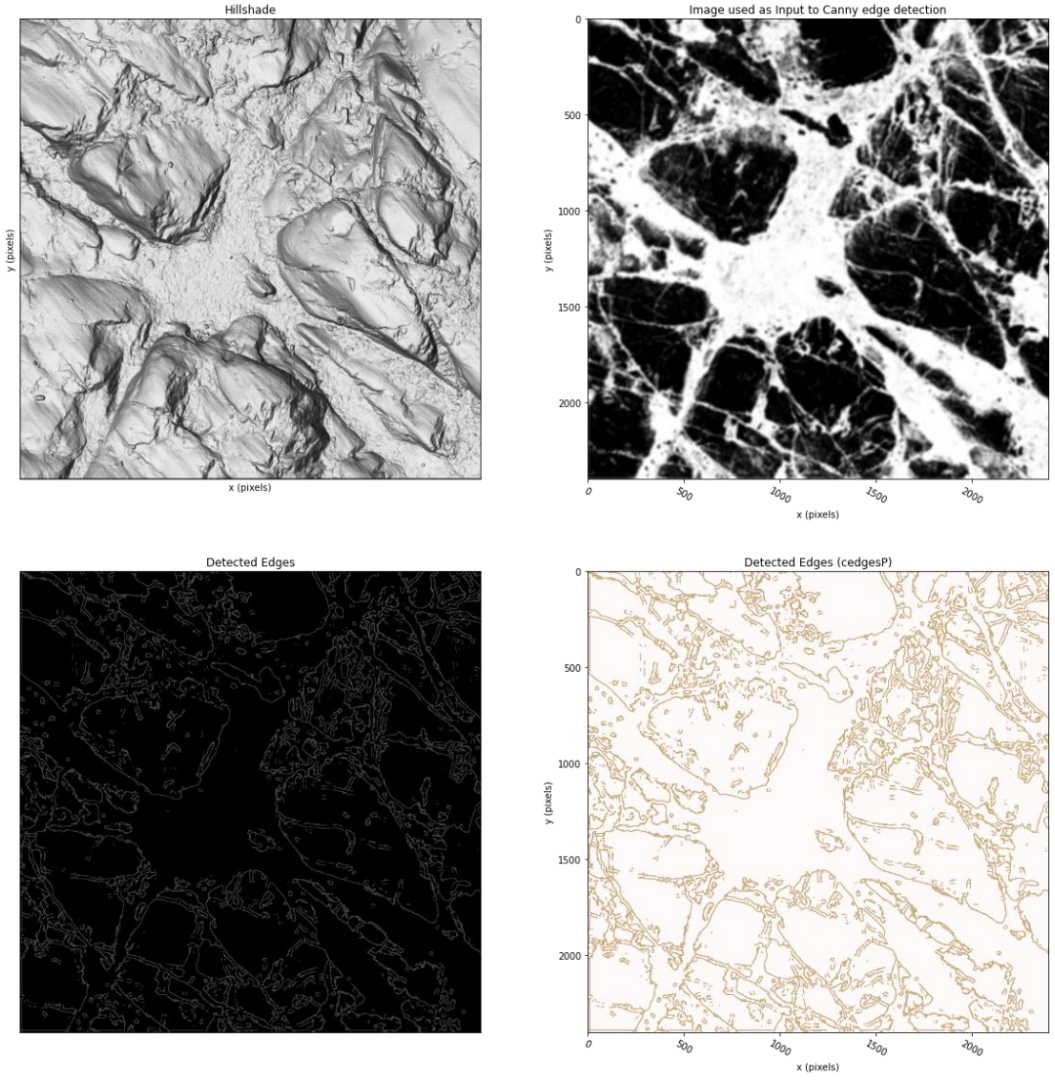

Fig. 12: Output of the Canny edge detection using the pre-processed orthophoto as input (top right).

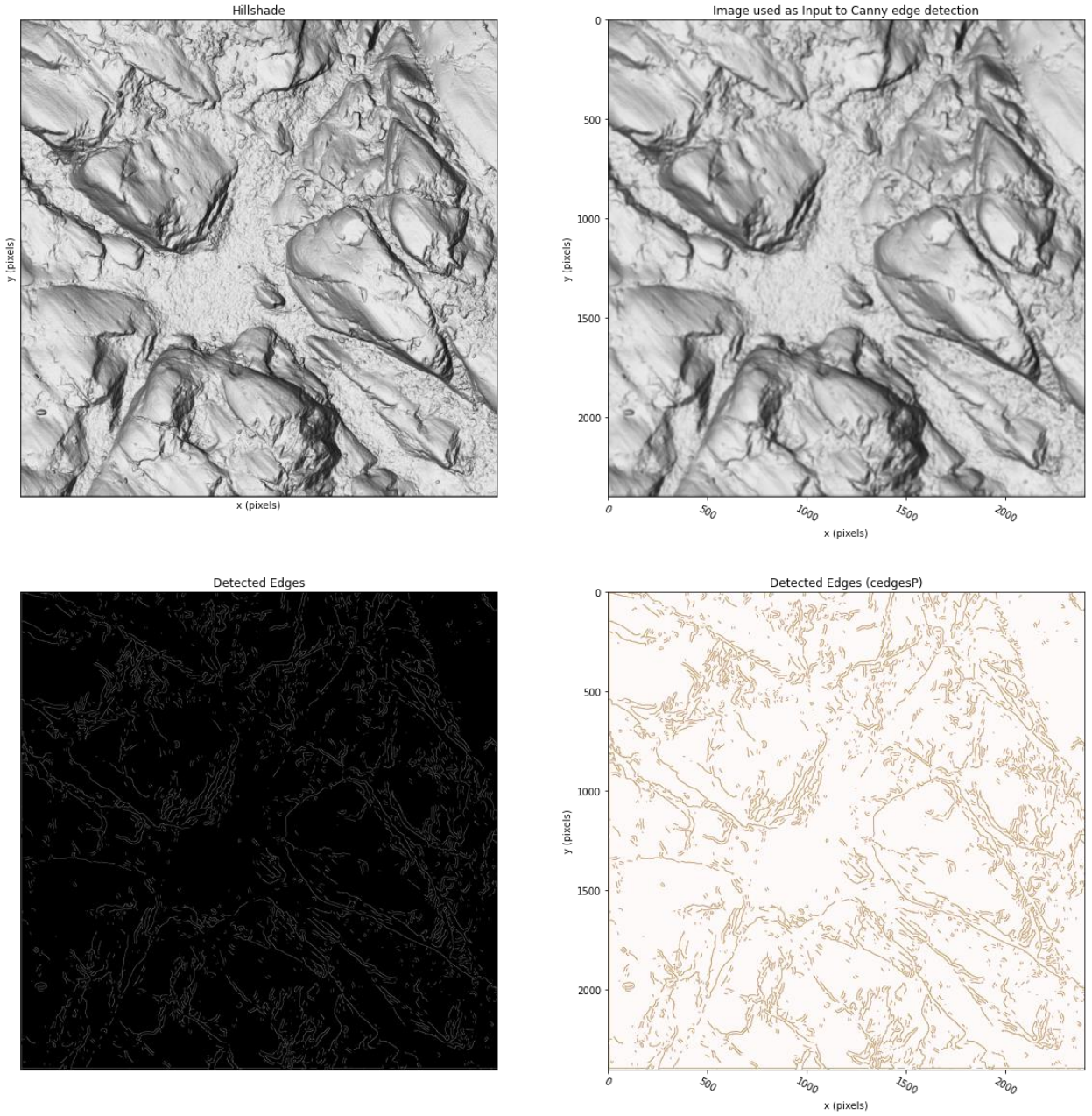

Fig. 13: Output of the Canny edge detection algorithm using the blurred hillshade as input (top right).

One example for the output created via Hough Transform is displayed in Figure 14. Many edges are well detected. However, there are also certain linear features that have not been successfully detected.

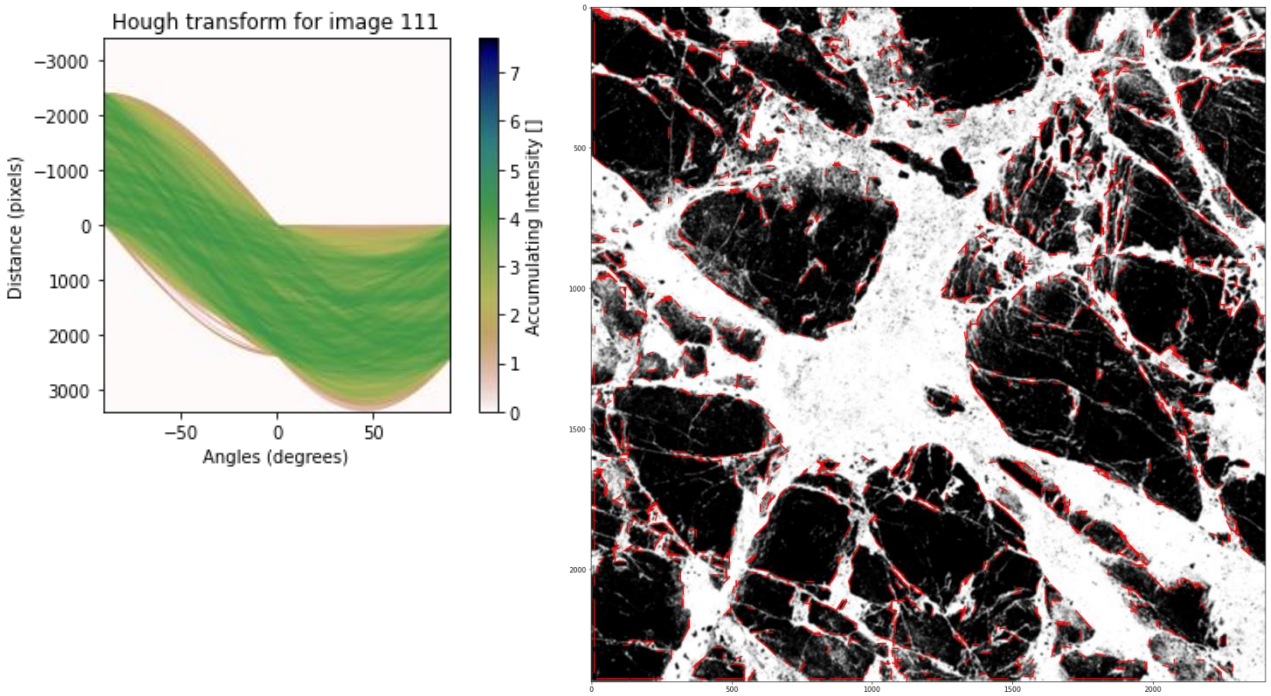

Fig. 14, Left: Hough Transform for image 111 (Table 2). A straight line in the (x, y) image space is represented by a single point with polar coordinates (ρ, θ) in the Hough-domain. This point is given by a bundle of intersecting sine curves in the Hough space, which eventually form a peak in the Hough image. Right: Detected lines after Probabilistic Hough Transform. Red lines mark the detected lineaments when using the shown orthophoto as input.

Testing the performance of Canny edge detection & Hough Transform using different Input images: Jaccard Score as Performance indicator

For a given set of parameters for Canny & Hough Transform, we tested how the ouput (amount of lineaments etc.) changes with different input images. For that reason, we defined the following set of parameters for the algorithms:

Threshold 1 = 45 rho = 0.5 minLineLength = 20
Threshold 2 = 50 theta = 0.5 maxLineGap = 5

```
threshold1=45  #PLAY WITH THIS Parameter for optimal results

threshold2=50  #PLAY WITH THIS Parameter for optimal results

edges = cv.Canny(image=image, threshold1=threshold1, threshold2=threshold2, apertureSize=3, L2gradient=True ) # Canny Edge Detection

linesP = cv.HoughLinesP(edges,rho=0.5,theta=0.5*np.pi/180,threshold=10,minLineLength=20,maxLineGap=5) # Max line Gap very sensitive, used to reduce amount of lineaments and merge them

for x1,y1,x2,y2 in linesP[0]:

    cv.line(cedgesP,(x1,y1),(x2,y2),(0,255,0),2)
```

Furthermore, we defined a performance indicator, to measure how well the detection works when comparing it with the reference dataset (man-made picked lineaments from Geologist). We use the Jaccard Score to evaluate the performance of the output.
"The Jaccard index [1], or Jaccard similarity coefficient, defined as the size of the intersection divided by the size of the union of two label sets, is used to compare set of predicted labels for a sample to the corresponding set of labels in y_true.
'micro': Calculate metrics globally by counting the total true positives, false negatives and false positives.
'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance"

(from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_score.html)

The Jaccard score allows us to compare two binary images with lineaments (predicted vs true) and how well these lineaments overlap with each other.

Challenges with Jaccard Score: The Jaccard score alone does not guarantee a good performance, since our lineaments cover only a tiny part of the overall image. Therefore, an image with no predicted lineaments still scores extremely high. An empty image (with no lineaments) leads to a Jaccard score of 0.96577! This is better than any prediction. Hence, there is a tendency for the score to perform better the less lineaments are predicted. It follows, that it can be only used as qualitative measure and it is very important to additionally aim for:

a) a high number of lineaments
b) a high number of pixels with lineaments detected
c) check the output visually (Fig. 15) and compare it to the true lineaments

Table 1 gives a list of all different inputs that has been tested including the performance. Figure 15 shows a selection of figures comparing the predicted to the true lineaments with the hillshade superimposed. Image names in Fig. 15 refer to names in Table 1.

Among the orthophotos as Input, Image 5 seems to perform best, despite having relatively low number of detected lineaments. Other images show more lineaments, but they some to scatter a lot and are not really related to true edges (e.g. Image 1, 3, 8 in Figure 15).

Among the hillshade images as Input, we only show part of all tests in Table 1. Overall, the Image 11 shows a good compromise between a high Jaccard Score and a reasonable amount of lineaments detected. Therefore, Image 5 and Image 11 are used for further optimization. The initially set parameters:

Threshold 1 = 45	rho = 0.5	minLineLength = 20
Threshold 2 = 50	theta = 0.5	maxLineGap = 5

are optimized and adapted in the next steps.

Table 1: Input performance analysis. List of input given to Canny edge detection showing Jaccard score for comparison of prediction with reference dataset. Best performing Input is marked in bold letters.

INPUT	Database	Explanation	Number lineaments	Number of pixels with lineament	Jaccard Score "macro"
Ref. data by Geologist	Orthophoto	Hand-picked from Orthophoto	4689	66016	
Image 1	Orthophoto	Orthophoto with increased contrast	3702	72906	0.9353
Image 2	Orthophoto	Inverse gray image with increased contrast & 5x5 gaussian blur	13157	216706	0.8685
Image 3	Orthophoto	Inverse gray image with increased contrast & 9x9 gaussian blur	4991	95844	0.9248
Image 4	Orthophoto	Inverse gray image with increased contrast & 15x15 gaussian blur	917	24086	0.9576
Image 5	**Orthophoto**	**Inverse gray image with increased contrast & 21x21 gaussian blur**	**836**	**22738**	**0.9583**
Image 6	Orthophoto	Blurred gradient image	2318	49350	0.9454
Image 7	Orthophoto	Sobel in x- & y- direction	42998	644517	0.6601
Image 8	Orthophoto	Sobel in x- & y- direction with Gaussian blur	17167	309516	0.8222
Image 9	Orthophoto	CNN model with filter convolution	629456	45989	0.6675
Image 10	Orthophoto	CNN model with filter convolution & Gaussian Blur	10410	190531	0.8804
Image 11	**Hillshade**	**Hillshade & 15x15 gaussian blur**	**505**	**17171**	**0.9606**
Image 12	Hillshade	Hillshade & 21x21 gaussian blur	423	14238	0.96205

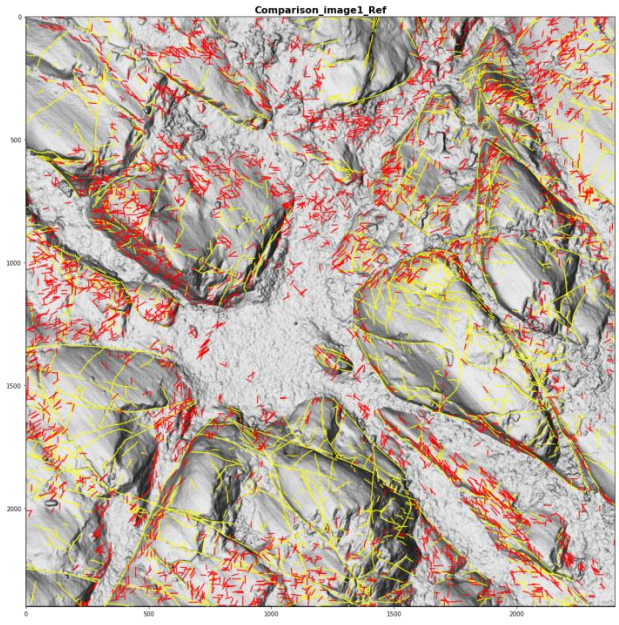

Image 1

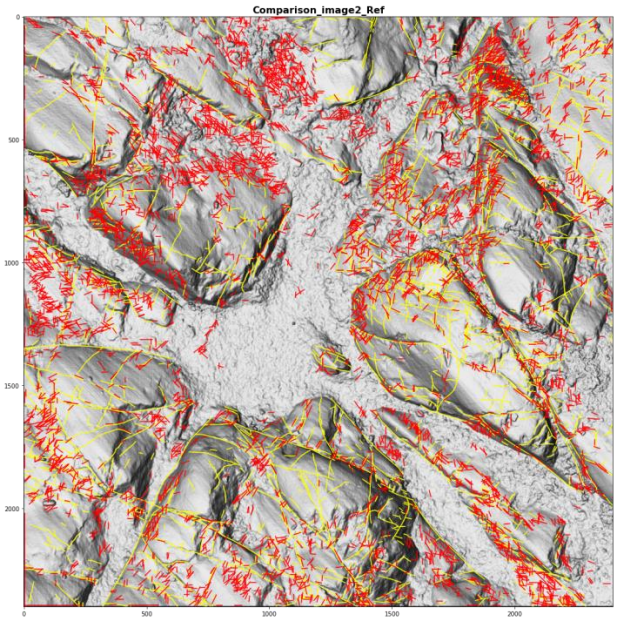

Image 3

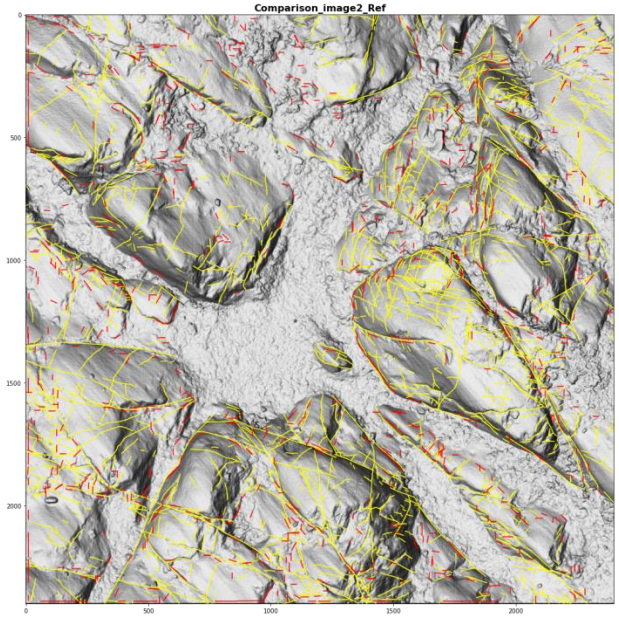

Image 5

Image 8

Figure continued on next page

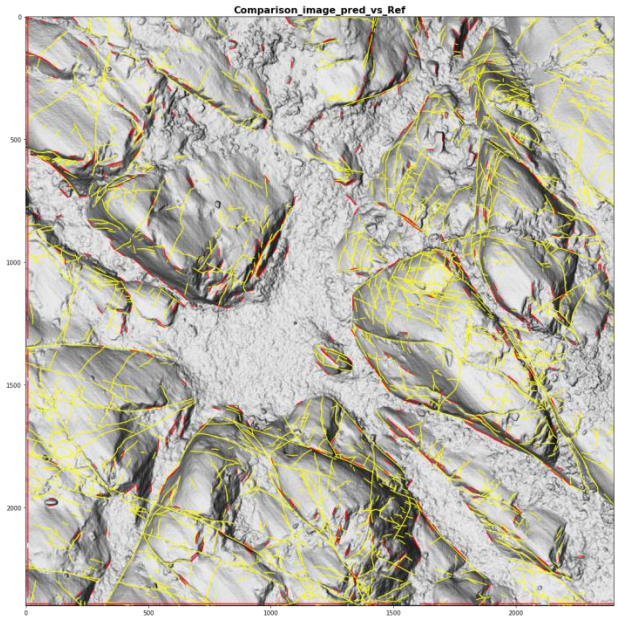

Image 11

Fig. 15: Comparison between the predicted (red) and the true (yellow) lineaments. Only a selection of results is shown. The image number refers to the input name in Table 1. The background shows the multi-directional hillshade.

Image 5 and 11 from Table 1 are further optimized varying the previously mentioned input parameters for the Canny edge detection and Hough Transform. The full list of all parameter setups tested is shown in Table 12. Please note, the Jaccard Score uses now “micro” as average, which accounts for inbalances.

The testing of various parameter sets indicates that the best performance is obtained for image 61 (Table 2), which was based on image 5 from Table 1 as input. Investigating the performance of the hillshade as input (image 11), we observe that image 113 shows a combination of parameters with good results. Some of the results are shown in Fig. 16 for visual inspection. It is important to note, that at the border of the image (picture frame), lineaments are detected as well. This is erroneous and we will have to remove those lineaments at a later stage (Goal C- Data Analysis).

As a final step of this subchapter, the detected lineaments in this section with x1, y1, x2, y2 – coordinates, are then stored in .csv - files and are exported to *Google Drive*.

In order to evaluate the output of the python-based lineament detection, we show in Table 3 some key numbers of the output created with the matlab-based algorithm. We show the results for two different inputs (multi-directional hillshade and orthophoto), and also list the mean direction (mean of angle) and median of direction of the detected lineaments. There were systematically more lineaments detected using matlab, when comparing to Table 2, which is mostly due to the minimum line length set in python. The matlab-based algorithm uses no minimum line length, which means that many short lineaments are included in Table 3. Furthermore, the tensor voting algorithm helps the Hough Transform algorithm to achieve better results.

Table 4 shows a small extent of the compiled lineament data table that is used in part (C)-Data Analysis and (D)-Clustering. Additional parameters apart from x1, y1, x2, y2 are calculated as needed.

Table 2: Parameter set testing for Image 5 and 11 from Table 1. List of input & parameters given to Canny edge detection showing Jaccard score for the comparison of prediction vs reference dataset. Best performing settings are marked in bold letters. Threshold for Canny above 130 leads to no detections.

INPUT	Data Input	Thresholds for Canny	Threshold Hough	Min Line Length	Max Line Gap	Num of lineaments	Num of pixels with lineament	Jaccard Score "micro"
Image 50	Ortho	100, 120	10	20	5	209	8897	0.9646
Image 51	Ortho	50, 120	10	20	5	228	9449	0.9643
Image 52	Ortho	50, 100	10	20	5	390	12968	0.9628
Image 53	Ortho	45, 50	10	20	5	836	22738	0.9584
Image 54	Ortho	49, 50	10	20	5	815	21944	0.9584
Image 55	Ortho	20, 30	10	20	5	1767	41347	0.9499
Image 56	Ortho	45, 50	10	15	5	1447	29344	0.9554
Image 57	Ortho	49, 50	10	20	6	1126	28177	0.9559
Image 58	Ortho	49, 50	8	20	5	796	21964	0.9586
Image 59	Ortho	49, 50	8	18	5	1107	26283	0.9568
Image 60	Ortho	49, 50	10	22	7	1136	30190	0.9550
Image 61	**Ortho**	**49, 50**	**10**	**25**	**8**	**1000**	**29397**	**0.9554**
Image 62	Ortho	**49, 50**	10	30	10	966	32445	0.9539
Image 110	Hillsh	45, 50	10	20	5	505	17171	0.9606
Image 111	Hillsh	49, 50	10	20	5	511	17316	0.9606
Image 112	Hillsh	49, 50	10	25	12	1197	33823	0.9532
Image 113	**Hillsh**	**49, 50**	**10**	**15**	**8**	**1717**	**34154**	**0.9530**

Table 3: List of key numbers from the output based on the matlab detection algorithm. Negative mean angles are counted from the ordinate counter-clockwise.

Input: Multi-directional hillshade: Num of Lineaments: 2726 Avg length of lineaments in pixel: 23.221054 Total num of segments: 6330.059360 Num of lineaments removed: 1210 Mean of angle: -13.109483 This is 95 confidence of mean angle: 1.229663 Median: -16.699244	Input: Orthophoto: Num of Lineaments: 2465 Avg length of lineaments in pixel: 22.555573 Total num of segments: 5559.948647 Num of lineaments removed: 1038 Mean of angle: -8.569065 This is 95 confidence of mean angle: 1.505807 Median: -14.534455

Table 4: Compilation of lineament data from all input images. Only part of columns is shown. Physical 3D segment length is calculated using DEM information.

	x1	y1	x2	y2	x mid	y mid	2D segment length	x start phys	y start phys	x end phys	y end phys	x mid phys
0	2080.0	2355.0	2076.0	2399.0	2078.0	2377.0	44.18	17.333333	19.625000	17.300000	19.991667	17.316667
1	719.0	2391.0	741.0	2399.0	730.0	2395.0	23.41	5.991667	19.925000	6.175000	19.991667	6.083333
2	390.0	2392.0	418.0	2399.0	404.0	2395.5	28.86	3.250000	19.933333	3.483333	19.991667	3.366667
3	606.0	2371.0	624.0	2397.0	615.0	2384.0	31.62	5.050000	19.758333	5.200000	19.975000	5.125000
4	432.0	2387.0	435.0	2397.0	433.5	2392.0	10.44	3.600000	19.891667	3.625000	19.975000	3.612500

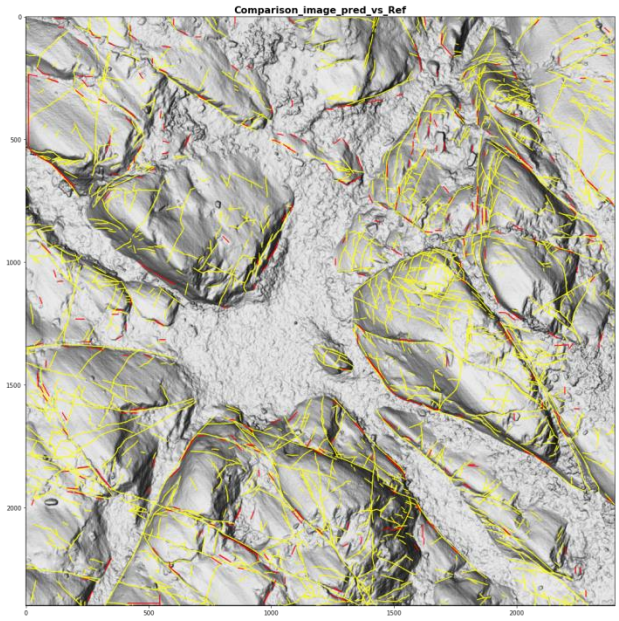

Image 51

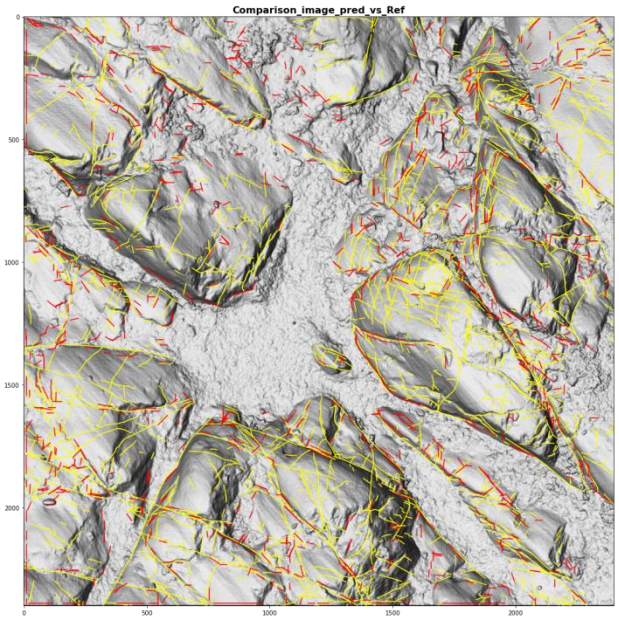

Image 61

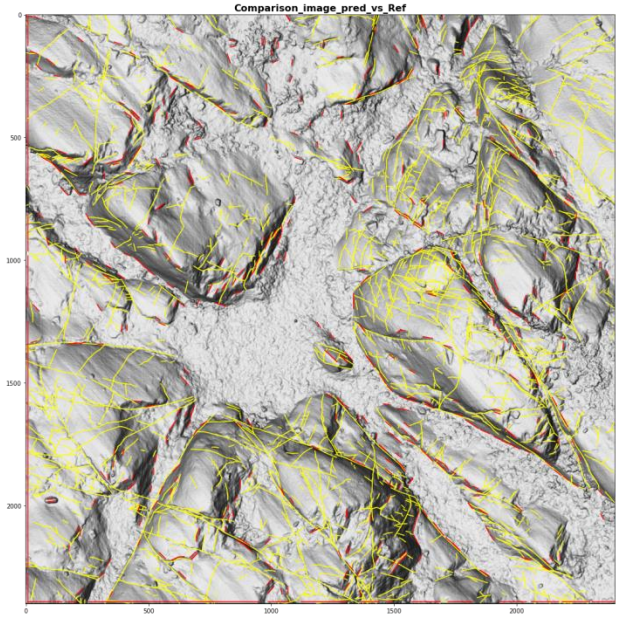

Image 111

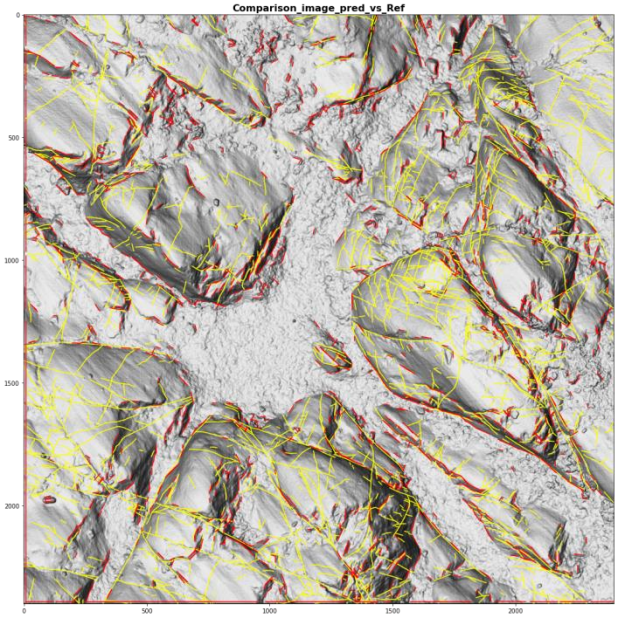

Image 113

Fig. 16: Comparison between the predicted (red) and the true (yellow) lineaments. Only a selection of results is shown. The image number refers to the input name in Table 2. The background shows the multi-directional hillshade.

3.3 Goal C – Data Analysis

This section of the thesis first reloads the exported csv files with lineaments form Goal B) and performs various ways to analyse and illustrate the data (histograms, Rose diagrams etc.). Beforehand, we remove some erroneous data and clean the dataset. All figures shown below use the following abbreviation in their titles:

RefGeol1: Reference dataset created by Geologist's interpretation of Orthophoto
Matlab_Ortho_slim: Lineaments from the matlab algorithm that used an Orthophoto as input
61: Lineament dataset that used an orthophoto as input with inverse gray imaging, contrast enhancement and 21x21 gaussian blurring as pre-processing (Table 2)
113: Lineament dataset that used Hillshade image with 15x15 gaussian blurring as input (Table 2)

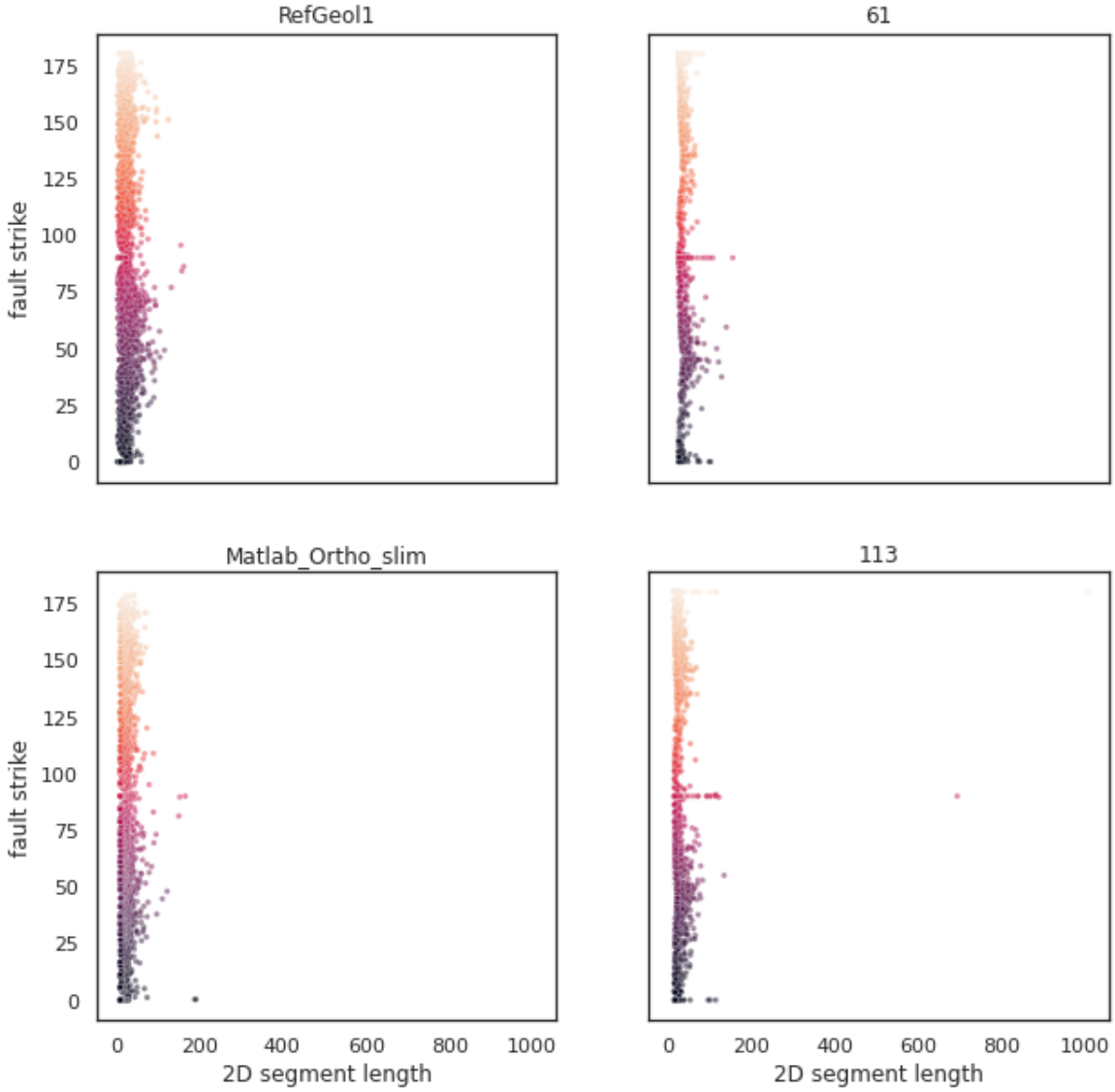

Fig. 17: 2D segment length vs fault strike. The data show clearly a very long segment length at around 90 degrees fault strike and at around 0 and 180 degree fault strike.

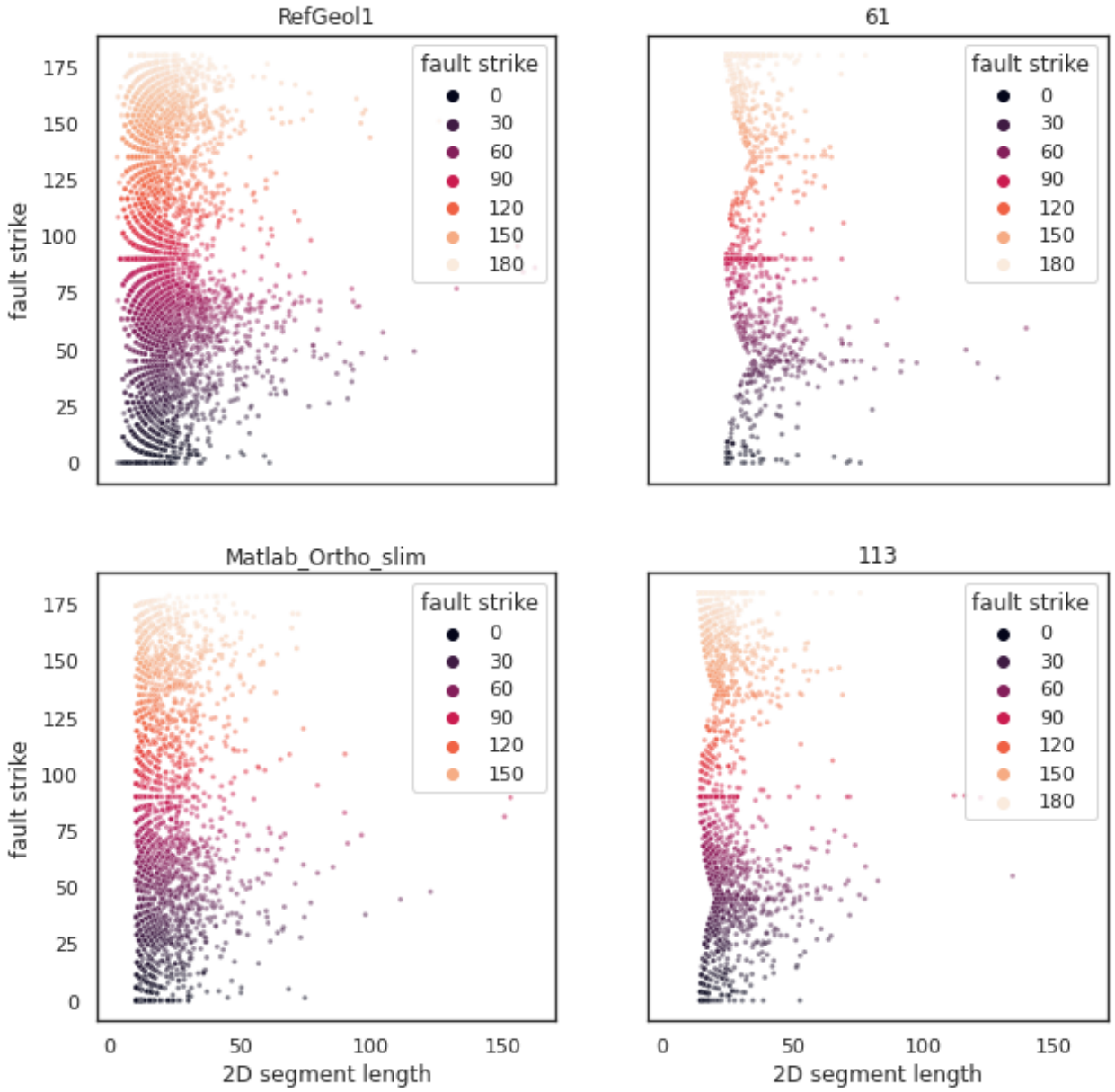

Fig. 18: 2D segment length vs fault strike. The cleaned data show no longer spikes at fault strikes around 0, 90, 180 degrees.

As a first step, all the lineament datasets had to be checked for systematic errors. Fig. 17 shows that spikes in segment lengths are systematically at 0, 90 and 180 degrees. This is related to the interpretation of image frames as edges. Therefore, with a simple for loop through all the datasets (except the reference dataset), we apply a condition that lineaments at 0, 90 and 180 degrees and longer than 80 pixels are removed from the data. The result of this cleaning is shown in Fig. 18.

As a next step for further data analysis, we plot the 4 datasets in various fashions shown in Figs. 19 – 21.

```
for i in range(1,4):
  df_tot_ind1 = df_list2[i][ (df_list2[i]['2D segment length'] >= 180) ].i
ndex # remove lineaments larger 180 pixels
  df_list2[i].drop(df_tot_ind1, inplace=True)
  df_tot_ind2 = df_list2[i][ (df_list2[i]['fault strike'] == 90) & (df_lis
t2[i]['2D segment length'] >= 80) ].index
  df_list2[i].drop(df_tot_ind2, inplace=True)
  df_tot_ind3 = df_list2[i][ (df_list2[i]['fault strike'] == 180) & (df_li
st2[i]['2D segment length'] >= 80) ].index
  df_list2[i].drop(df_tot_ind3, inplace=True)
  df_tot_ind4 = df_list2[i][ (df_list2[i]['fault strike'] == 0) & (df_list
2[i]['2D segment length'] >= 80) ].index
  df_list2[i].drop(df_tot_ind4, inplace=True)
```

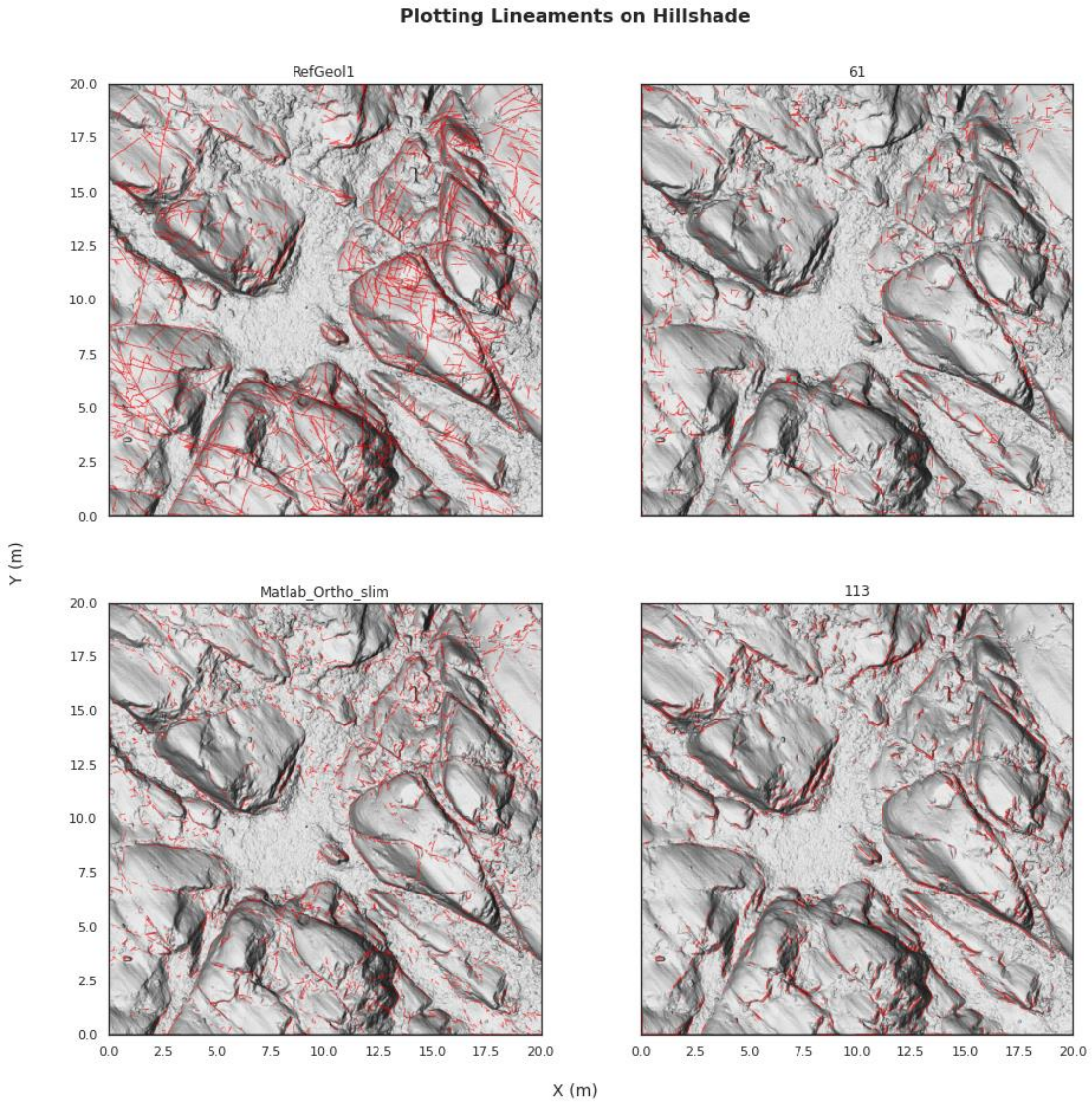

Fig. 19: Plotting all lineaments with the multi-directional hillshade as background.

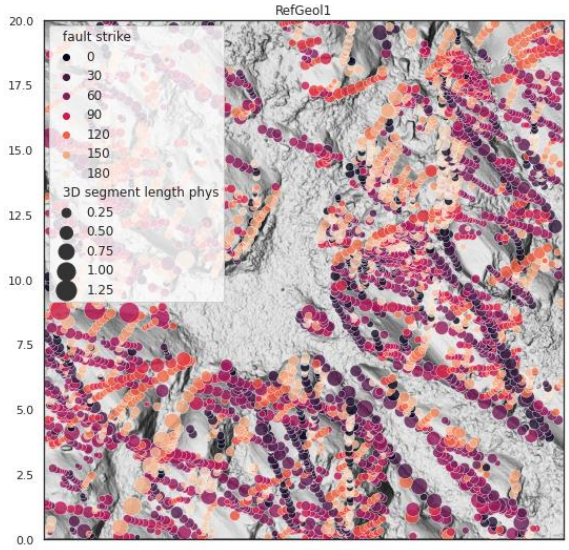

Fig. 20: Combining the information of physical 3D segment length with fault strike.

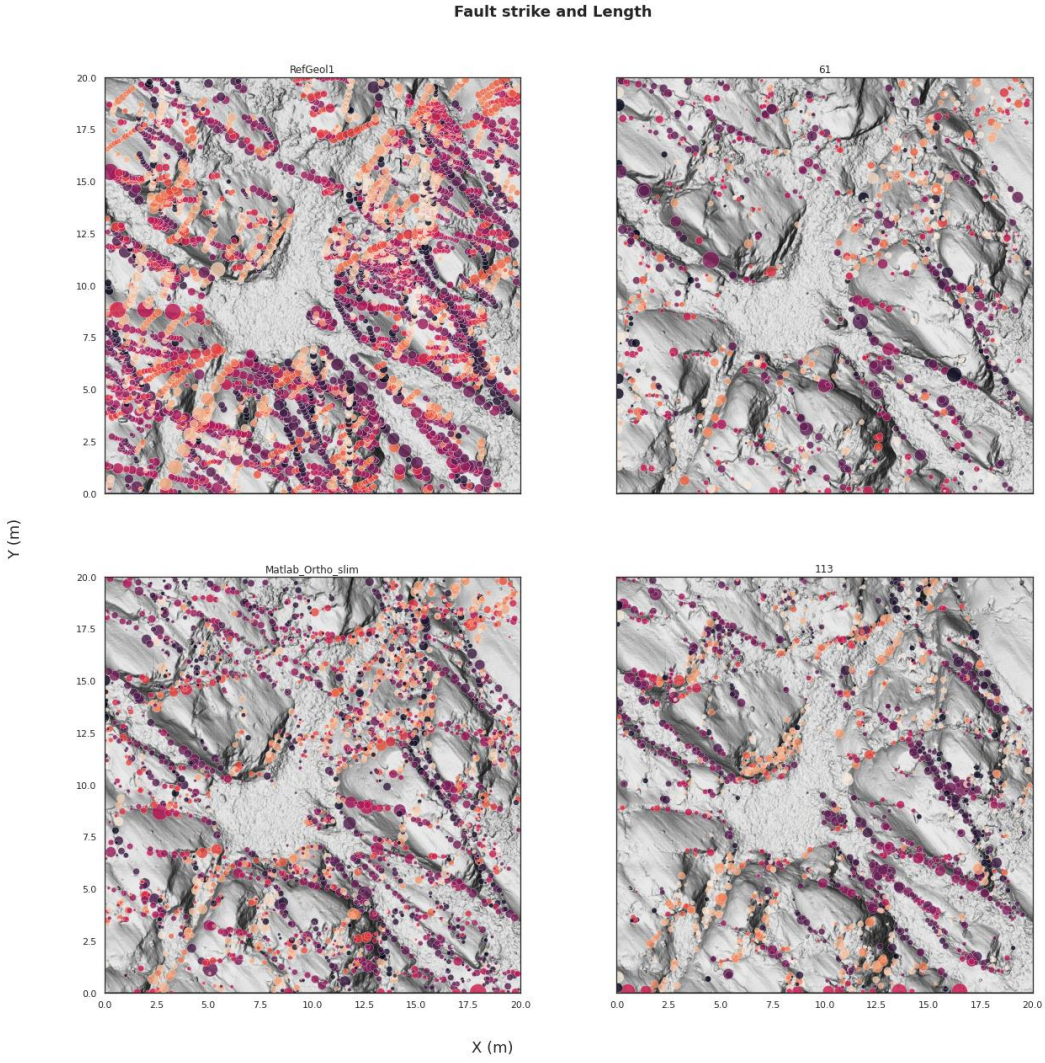

Fig. 21: Combining the information of physical 3D segment length with fault strike.

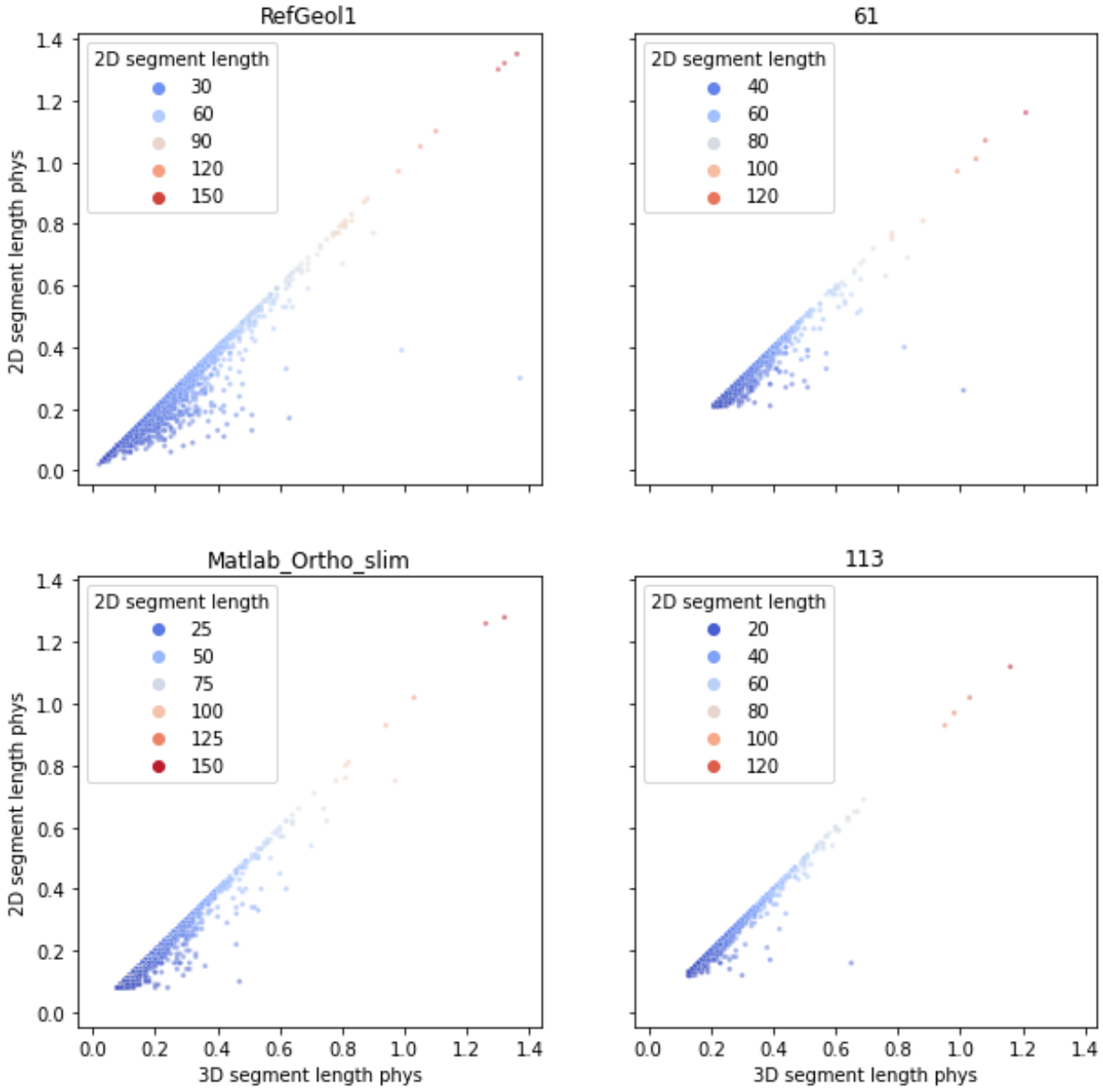

Fig. 22: Plotting physical 3D segment length vs physical 2D segment length.

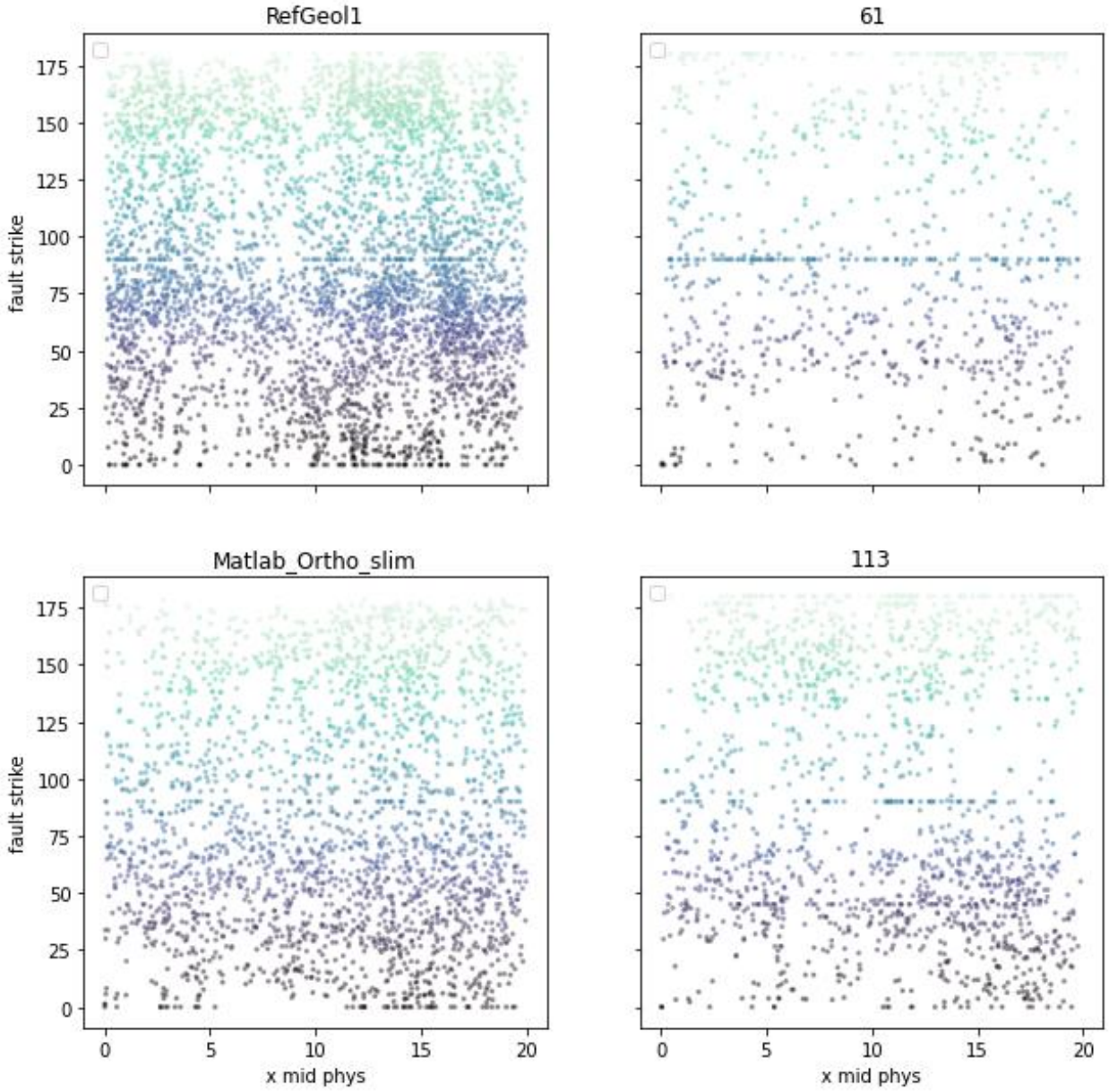

Fig. 23: Plotting physical midpoint (only x coordinate) vs fault strike.

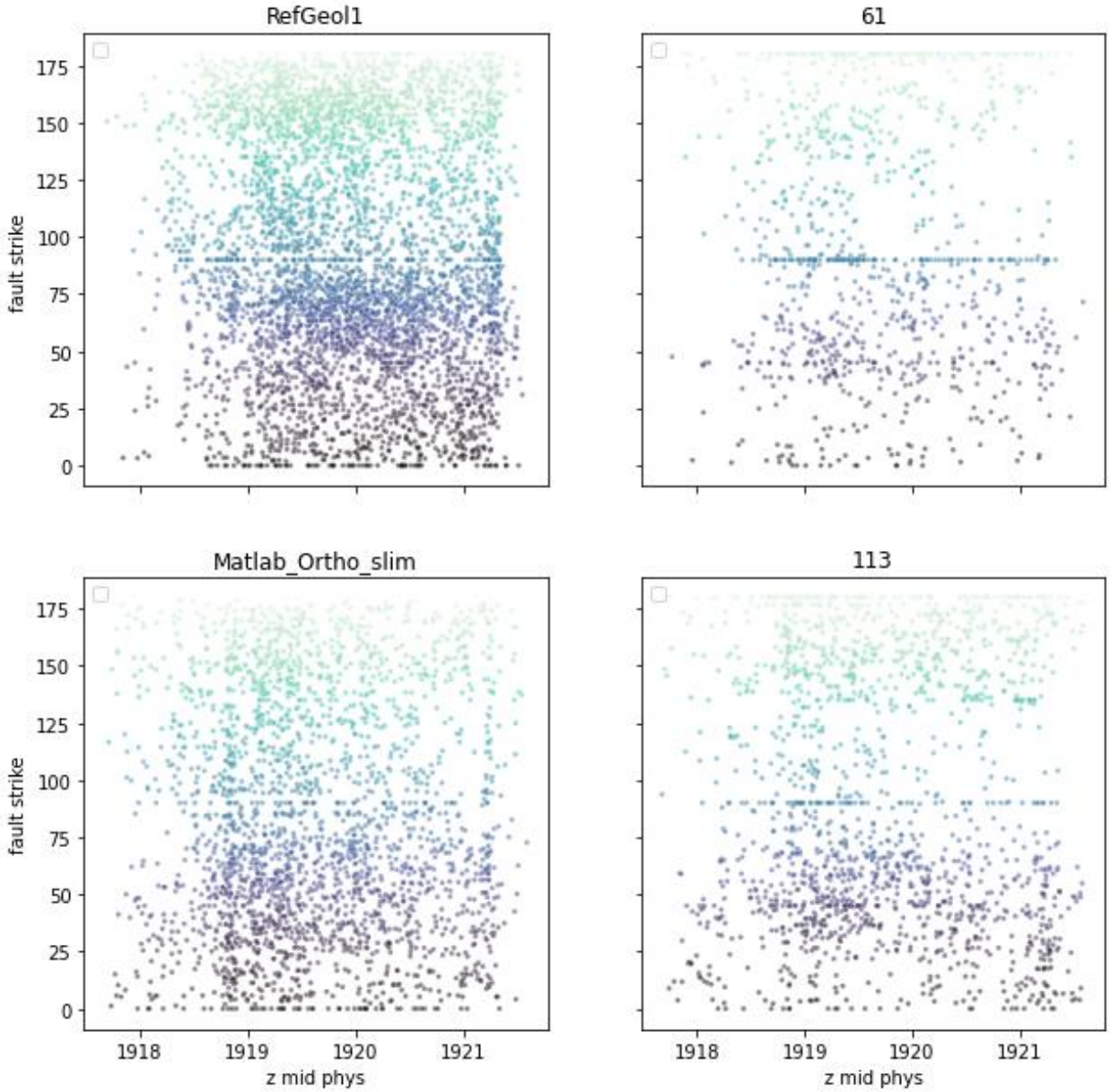

Fig. 24: Plotting physical midpoint (only z coordinate) vs fault strike.

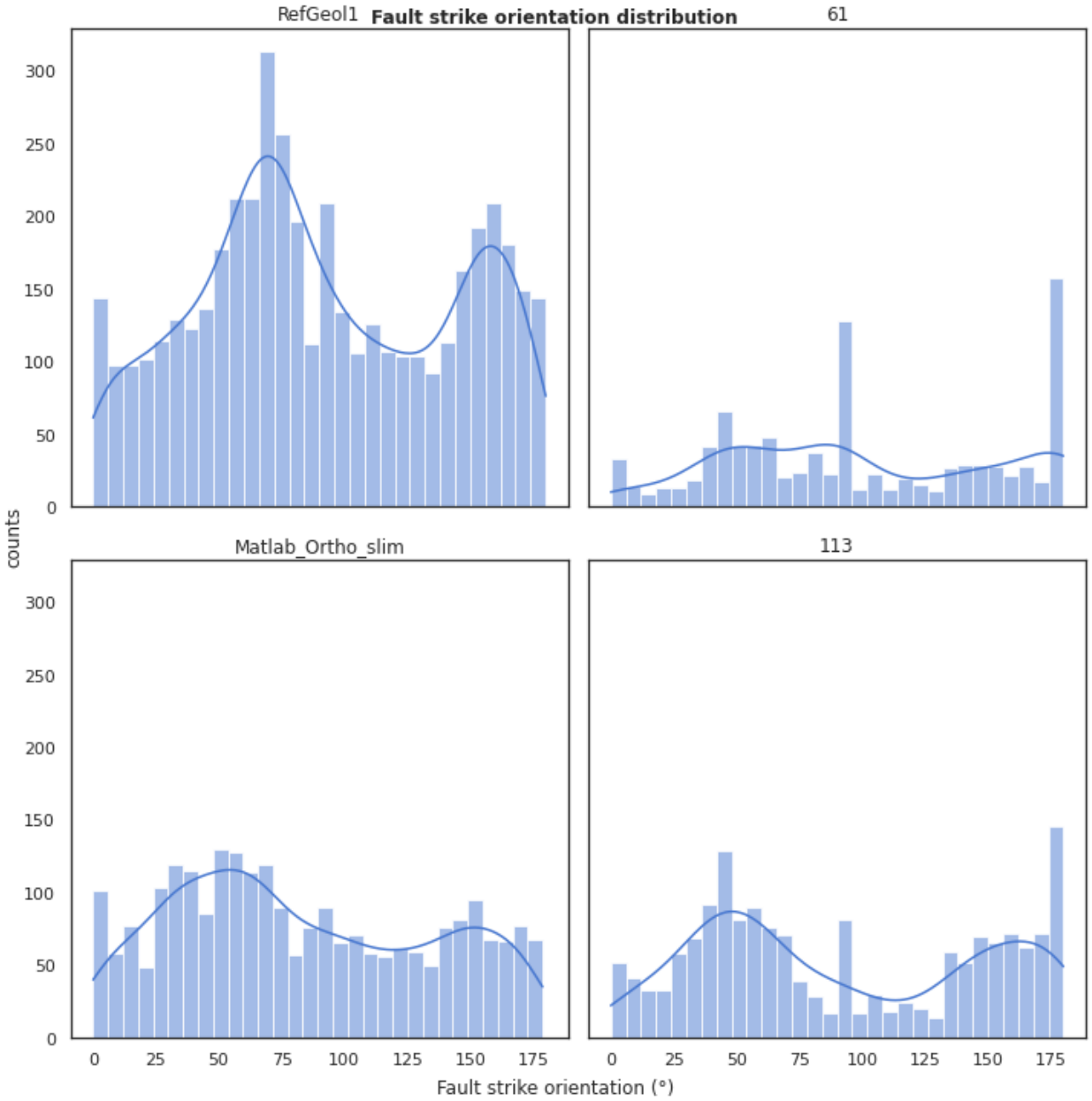

Fig. 25: Histogram of fault strike. Dataset 61 and 113 show some strong peaks at 90 degrees and 180 degrees, likely indicative that not all misinterpretations of the image frame have been removed.

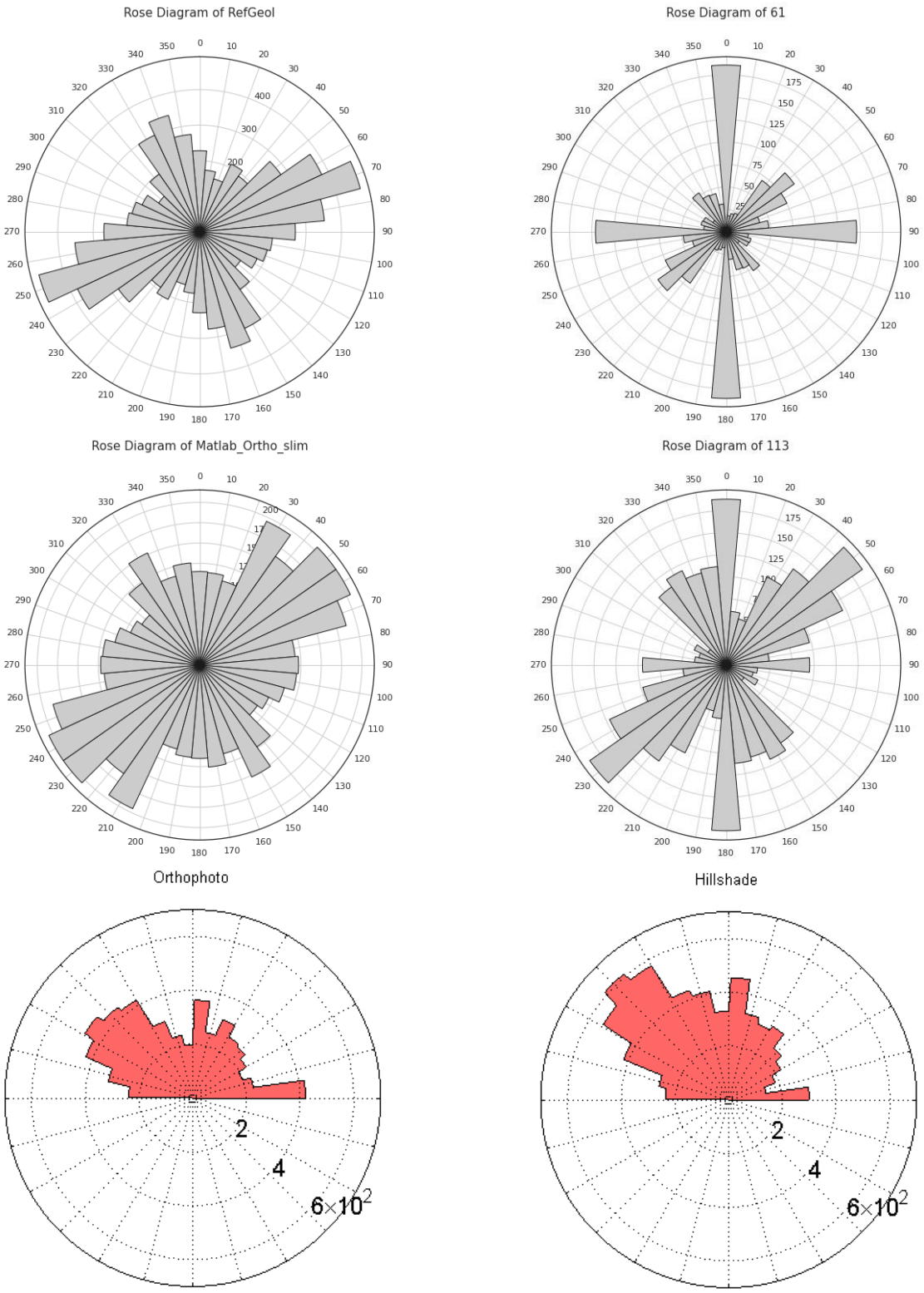

Fig. 26: Rose diagrams showing the orientation of the lineaments. Bin size was set to 10 degrees. The red diagrams are showing output from the matlab-based code and are reduced to the upper hemisphere. Inner circles indicate the total amount of lineaments detected. As noted in Fig. 25, dataset 61 and 113 show some dominant orientations for 90 and 180 degrees of fault strike, likely due to the image frame.

3.4 Goal D – Unsupervised machine learning: Clustering

This script reloads the exported csv files with lineaments from Goal B) and performs various ways to analyze and interpret and cluster the data with unsupervised machine learning algorithms (e.g. k-means etc.). Furthermore, we look at the uncertainty of the clustering predictions using a covariance matrix.

The list of available clustering algorithms is almost endless. We tested most of the clustering algorithms presented in Fig. 27. In this thesis, we show the results of a few of those tests. We focused on "k-means", "DBScan" and "Birch Clustering" algorithms. The python script additionally shows results for:

- Affinity propagation clustering
- Gaussian mixture clustering
- mean shift clustering

k-means

"The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified." (from: https://scikit-learn.org/stable/modules/clustering.html#k-means)

For k-means applications, the number of clusters is the primary parameter to be changed. Figs. 28-30 show some results how k-means puts the centroids for the various lineament datasets and number of clusters plotting 3D segment length against fault strike and the midpoint of the z component against fault strike.

DBSCAN

"DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

- *Eps: float, default=0.5. The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.*
- *Min_samples: int, default=5. The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself."*

(from: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html)

For DBSCAN applications, the "eps" and the "min_samples" are the primary parameters to be changed for best results. In Fig. 31, we investigate how DBSCAN performs using different sets of "eps" and "min_samples".

Birch Clustering

"It is a memory-efficient, online-learning algorithm provided as an alternative to MiniBatchKMeans. It constructs a tree data structure with the cluster centroids being read off the leaf. These can be either the final cluster centroids or can be provided as input to another clustering algorithm such as Agglomerative Clustering.

- *Threshold: float, default=0.5. The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold. Otherwise a new subcluster is started. Setting this value to be very low promotes splitting and vice-versa.*
- *n_clusters: int, instance of sklearn.cluster model, default=3. Number of clusters after the final clustering step, which treats the subclusters from the leaves"*

(from: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html)

Birch clustering uses two parameters as primary setting input, as presented above. In Fig. 32, we varied the two parameters and documented its effect on the clustering. It is important to note that the Birch Clustering algorithm performs very well with the linearly behaving data, when plotting 2D segment length against the 3D segment length. Other algorithms have difficulties assign clusters to such data.

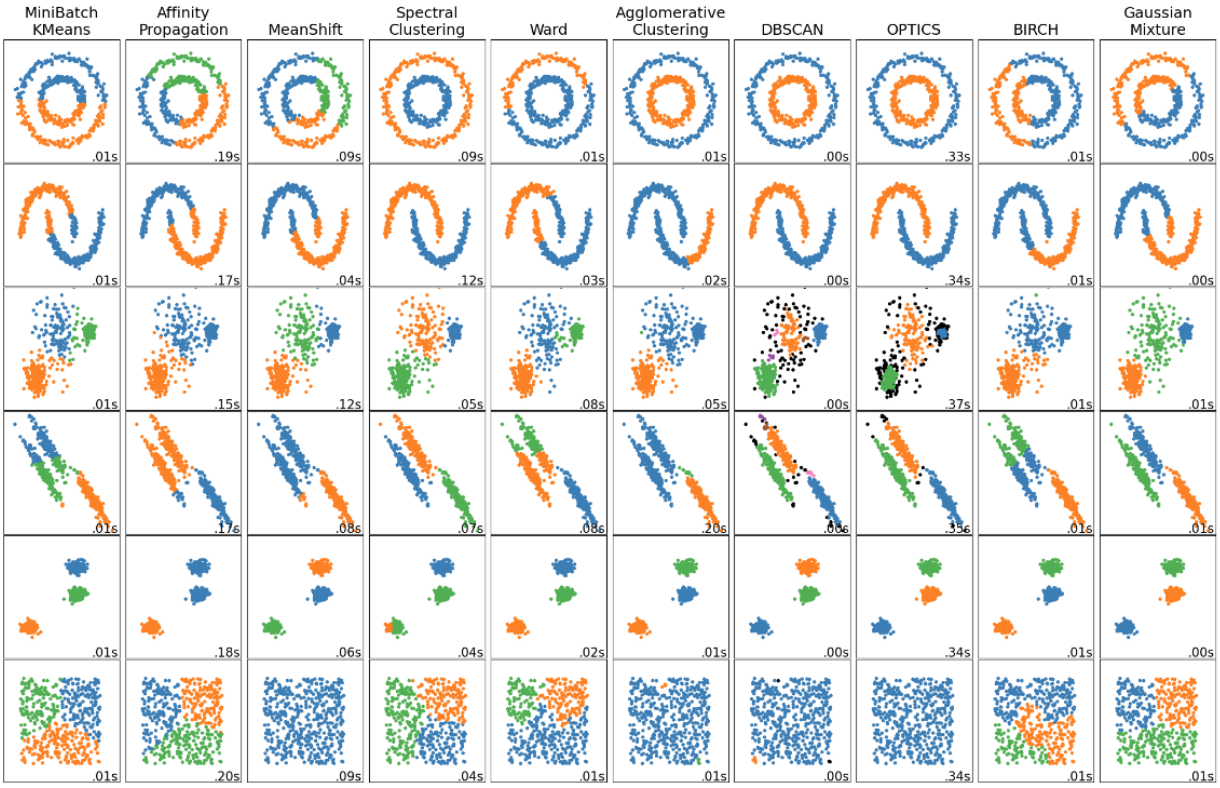

Fig. 27: Available algorithms for data clustering (from: https://scikit-learn.org/stable/modules/clustering.html)

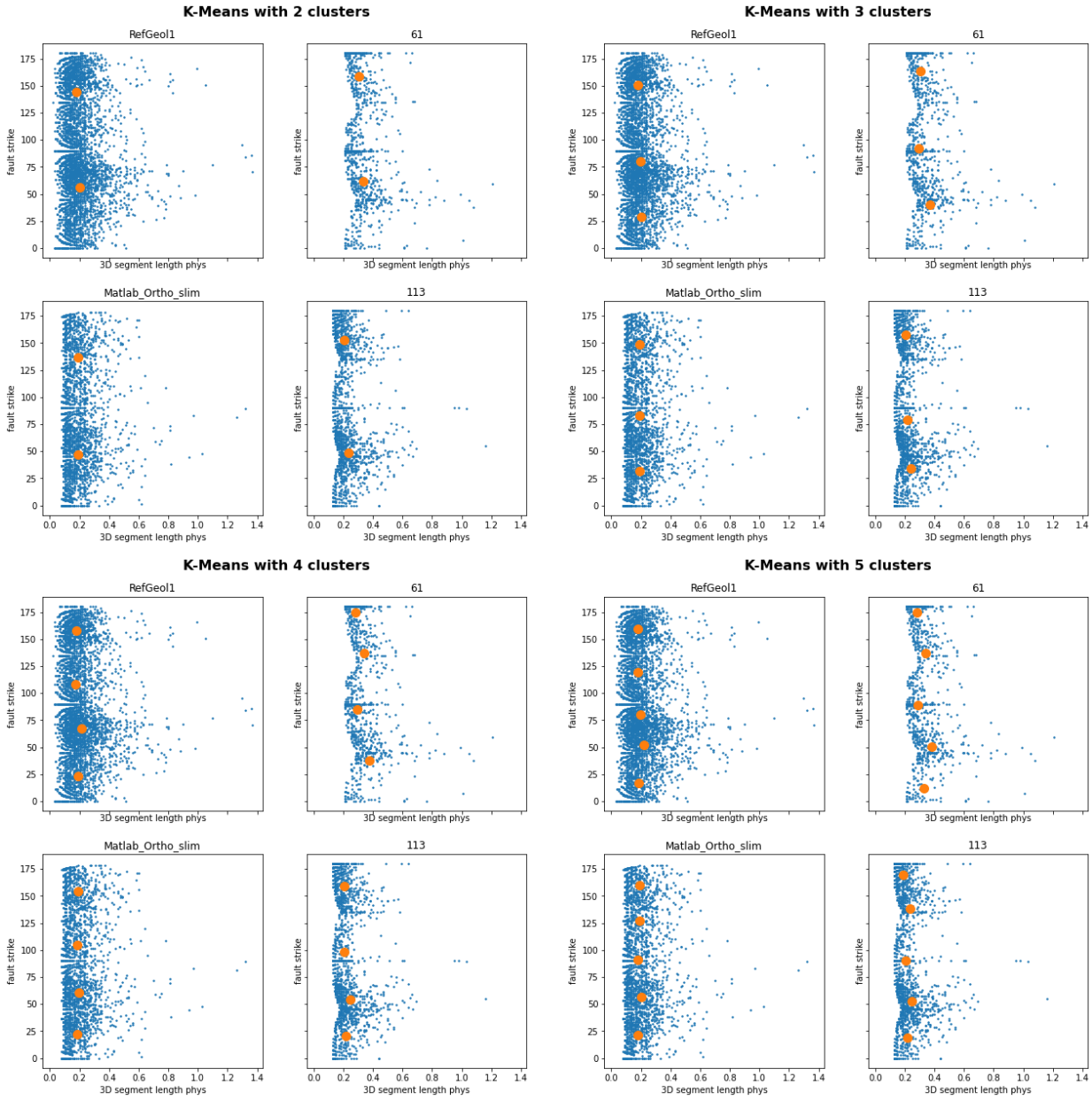

Fig. 28: 3D segment length vs fault strike with 2 to 5 cluster fittings. Orange dots mark the cluster centroids.

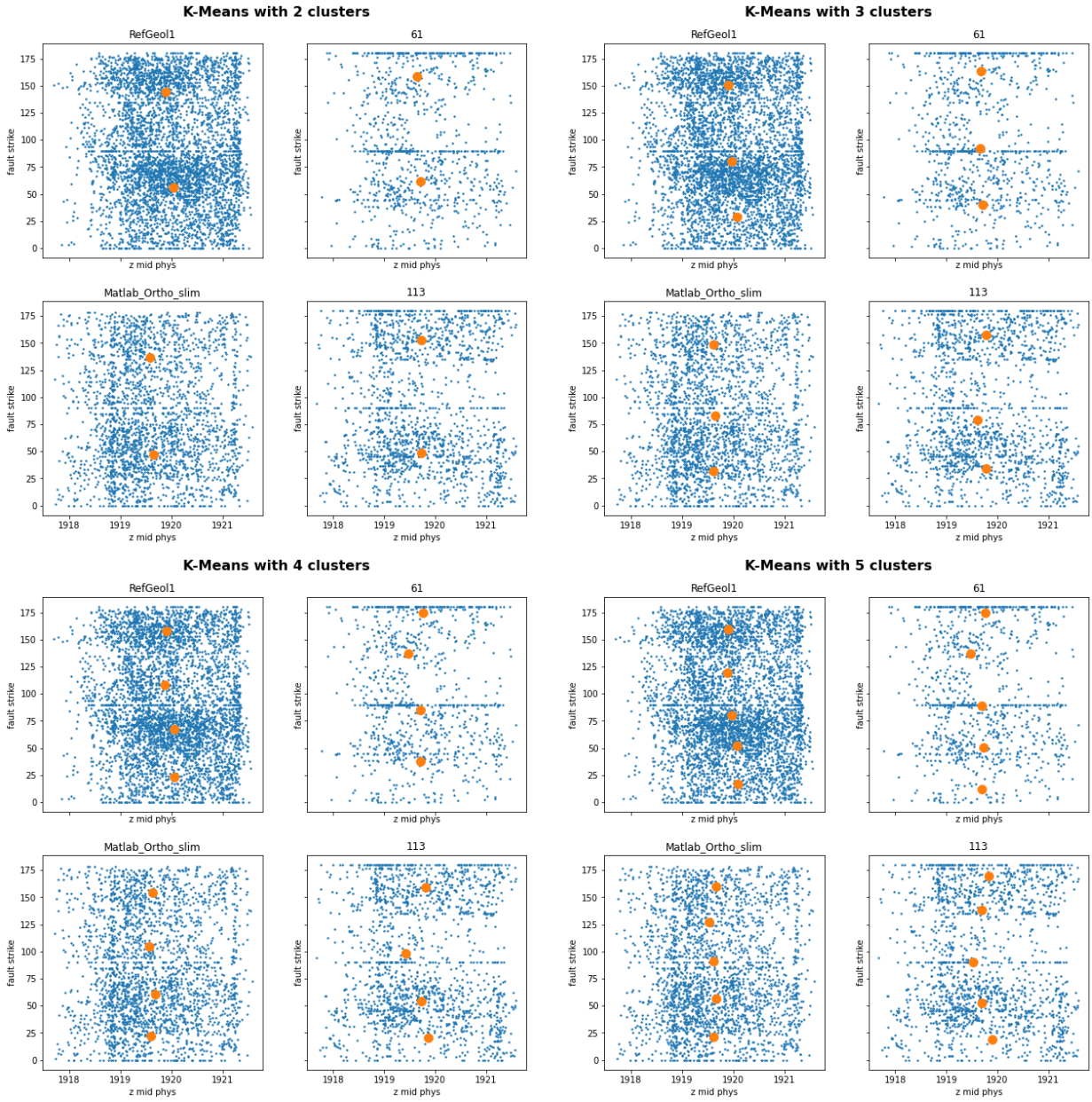

Fig. 29: Physical midpoint of z component vs fault strike with 2 to 5 cluster interpretations, using k-means. Orange dots mark the cluster centroids.

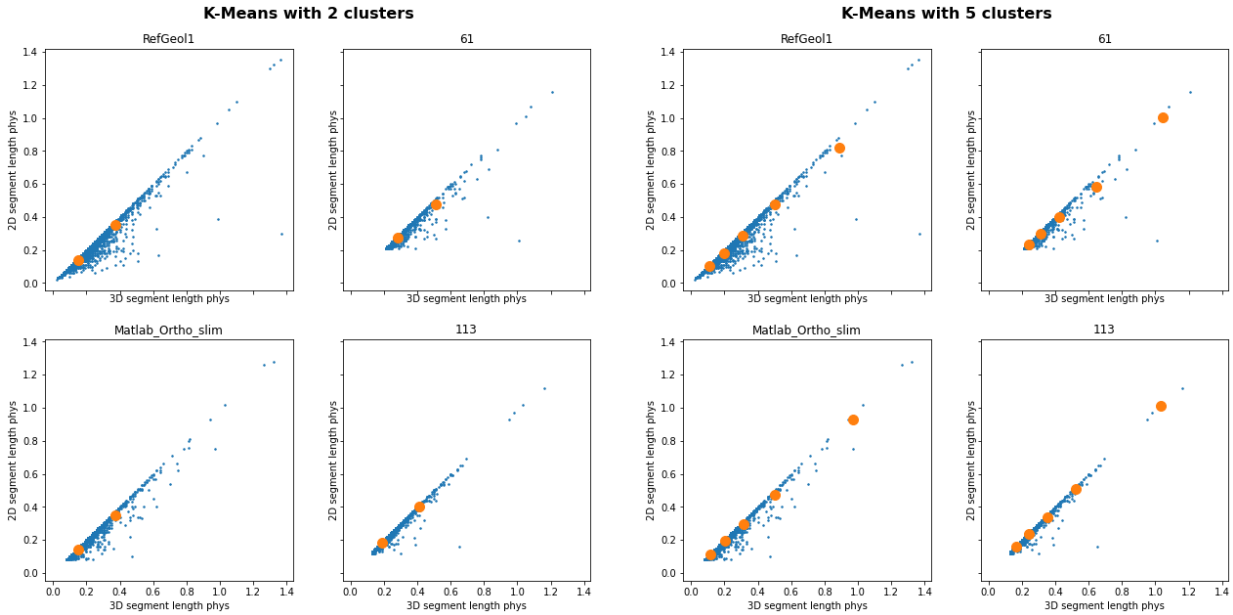

Fig. 30: Physical 3D segment length vs 2D segment length with 2 to 5 cluster interpretations, using k-means. The 3D segment lengths are slightly higher than the 2D segment lengths, since in 3D a topographic component is included. Orange dots mark the cluster centroids.

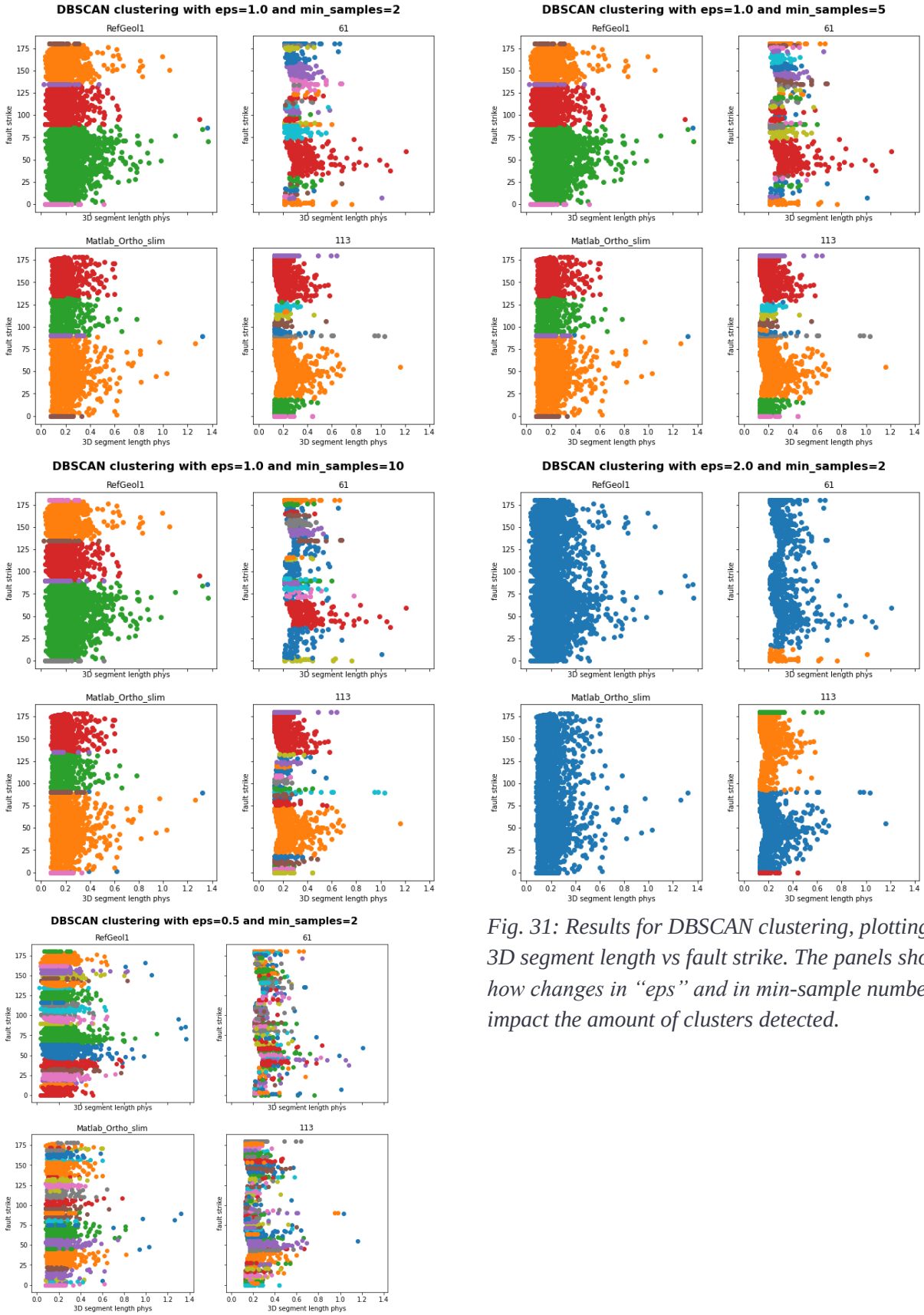

Fig. 31: Results for DBSCAN clustering, plotting 3D segment length vs fault strike. The panels show how changes in "eps" and in min-sample number impact the amount of clusters detected.

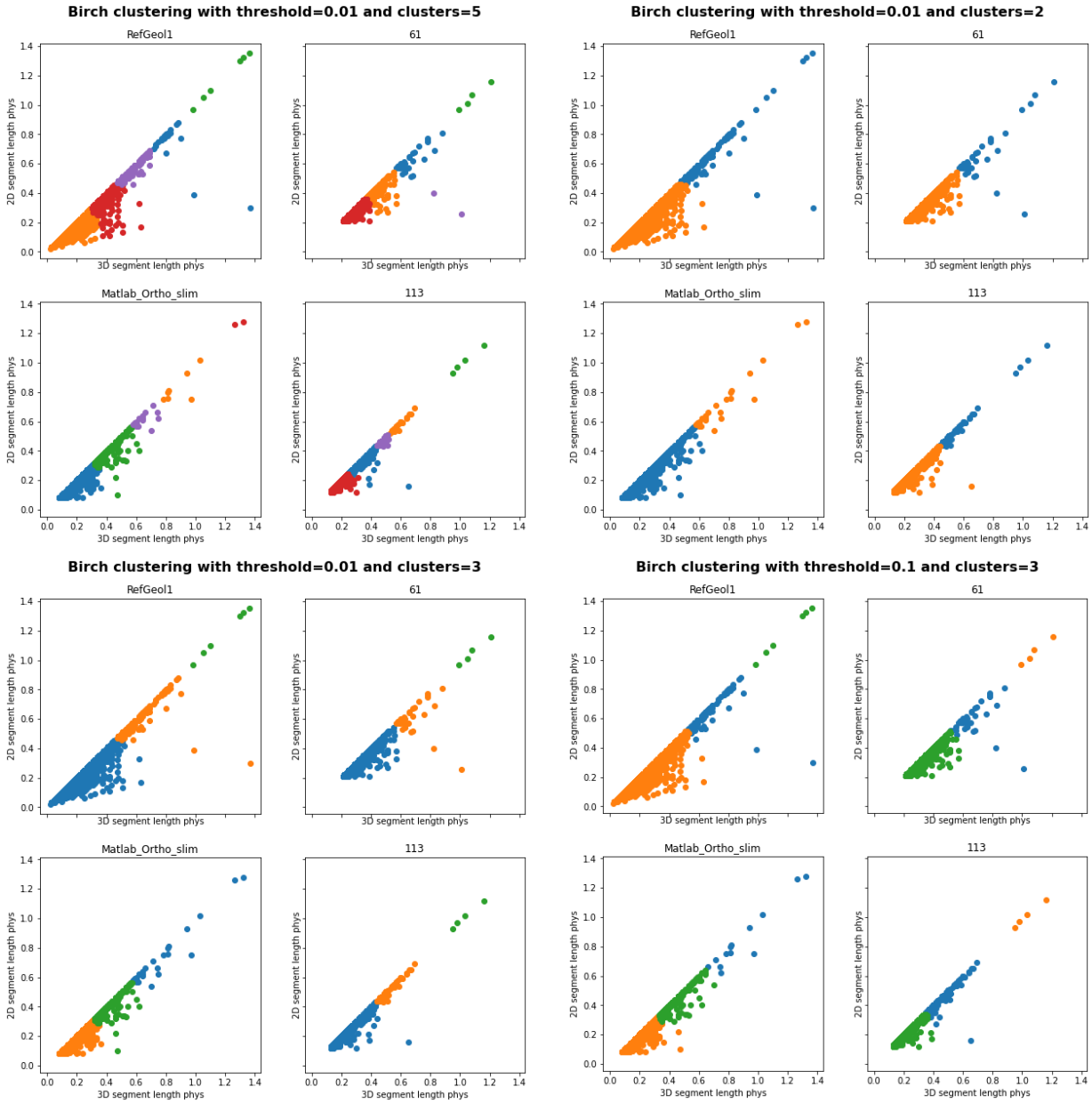

Fig. 32: Results for Birch clustering, plotting 3D segment length vs 2D segment length. The data does illustrate how the threshold and number of clusters impacts the grouping of data points.

The Silhouette Coefficient

"The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max(a, b). To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is 2 <= n_labels <= n_samples – 1"

(from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html)

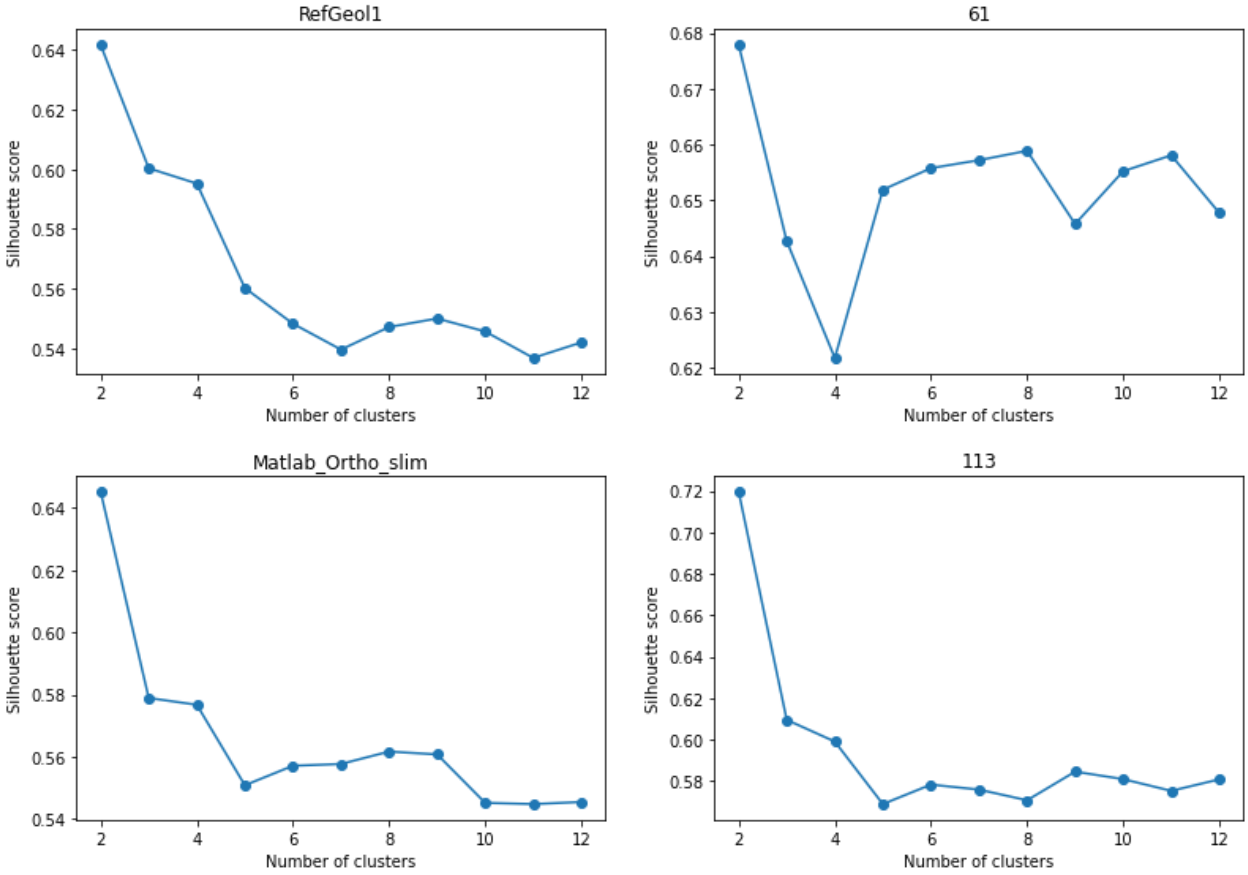

Fig. 33: Silhouette score for cluster numbers from 2 to 12, plotting physical 3D segment length vs fault strike.

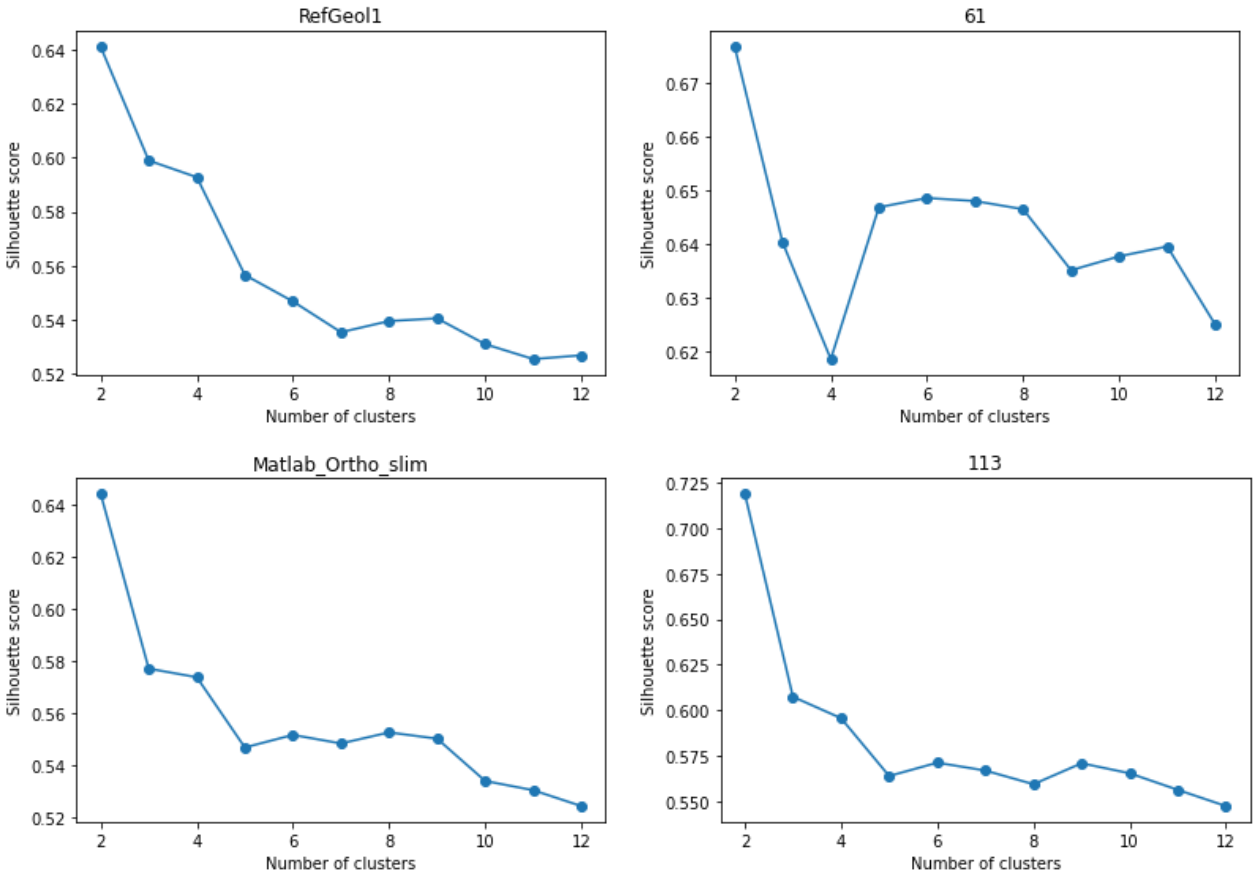

Fig. 34: Silhouette score for cluster numbers from 2 to 12, plotting physical midpoint of z component vs fault strike. Interestingly, the dataset 61 shows an increase in score for cluster numbers 5 to 8.

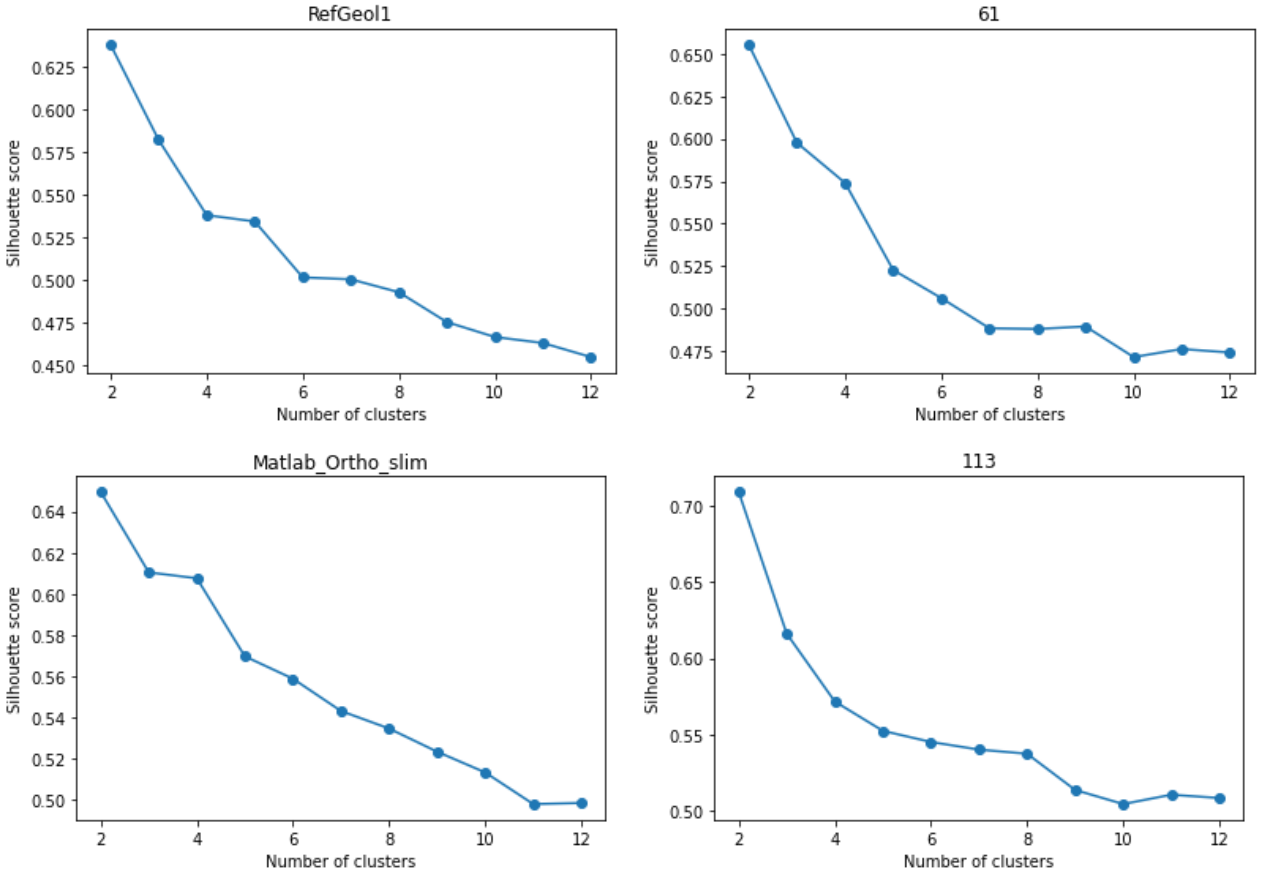

Fig. 35: Silhouette score for cluster numbers from 2 to 12, plotting 3D segment length vs 2D segment length.

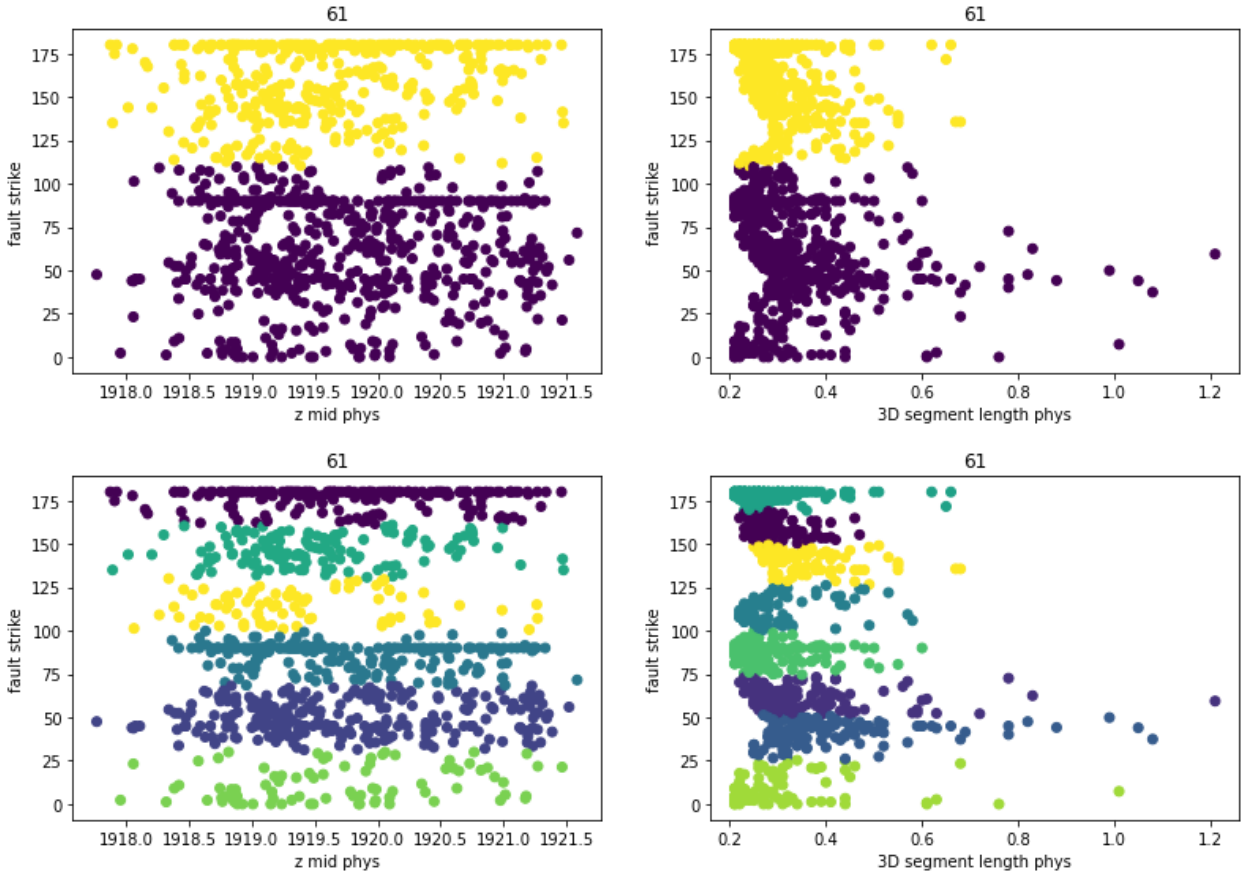

Fig. 36: Complementary figure for dataset 61 using k-means, showing the clustering (color-coded) for elevated Silhouette Scores at high cluster numbers in figs.33 and 34. Left: Clustering output for 2 and 6 clusters. Right: Clustering output for 2 and 8 clusters.

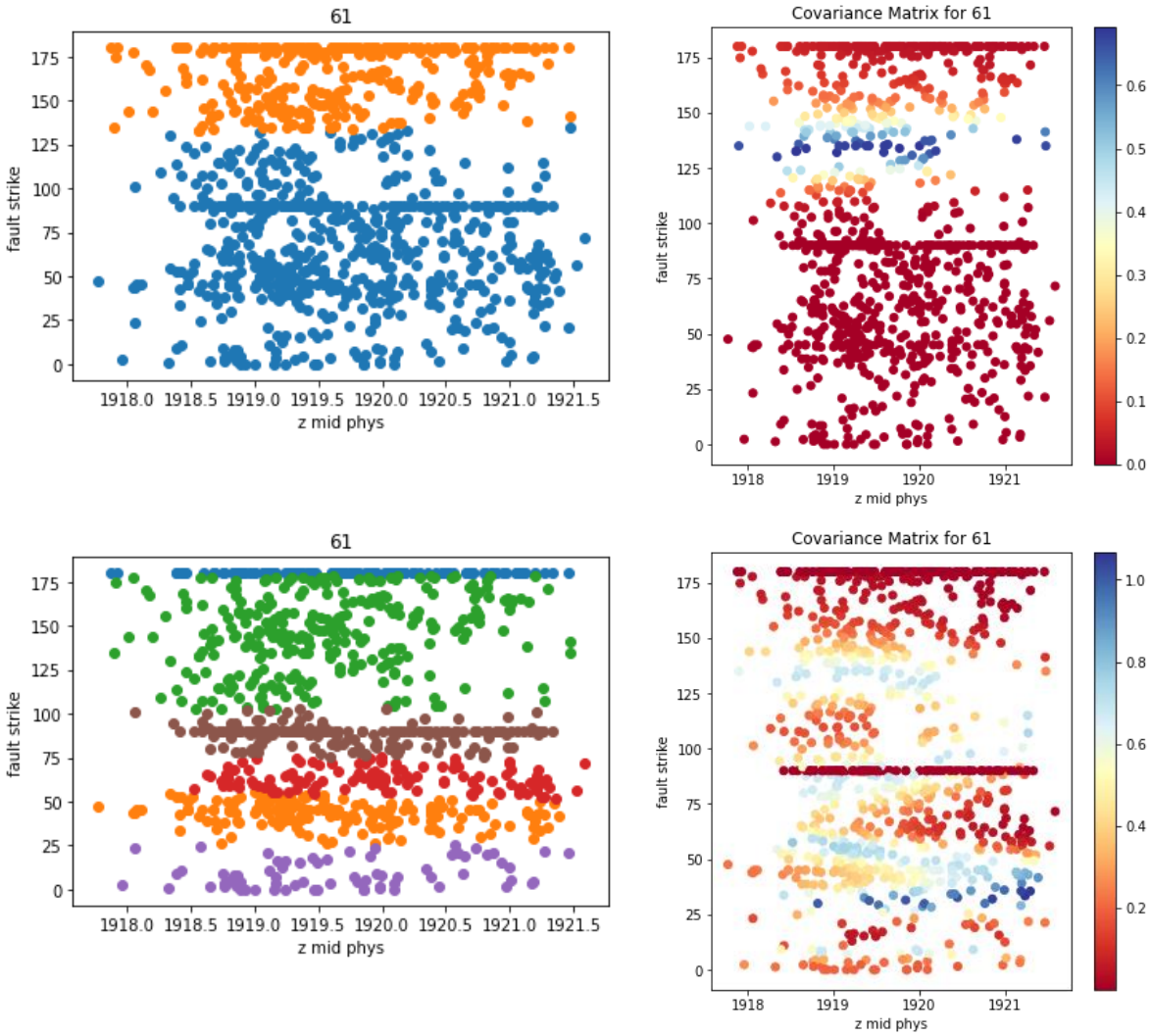

Fig. 37: Fitting for 2 and 6 clusters based on the Gaussian Mixture fitting algorithm (left).The corresponding covariance matrix is shown on the right. Overall, the uncertainty is lower for 2 clusters than for 6 clusters.

The Covariance Matrix

The covariance matrix shows the probability distribution over the label, apart from only predicting the label. Each point in the covariance matrix reflects the uncertainty of its prediction. It is important to note, that the covariance matrix is based on the Gaussian Mixture clustering. Fig. 37 shows fitting for 2 and 6 clusters based on the Gaussian Mixture fitting algorithm and additionally the corresponding covariance matrix. Overall, the uncertainty for two clusters is lower than for 6 clusters, as expected and indicated by the Silhouette Score in Fig. 34.

Fig. 38 shows a list of covariance matrices for 2 clusters, when plotting the 3D segment length against the fault strike. More results of covariance matrices can be taken from the python script D).

The python code for the covariance matrix is shown below.

```
i=0
n_components = 2


for i in range (4):
  points=df_3Dslp_fs[i] ## INPUT
  plt.figure(figsize=(6,6))
  clf = GaussianMixture(n_components=n_components, covariance_type='full')
  clf.fit(points)


  cluster_labels_prob=clf.predict_proba(points)

  entropies=[]
  for point in range(len(cluster_labels_prob)):
      entropies.append(entropy(cluster_labels_prob[point]))

# 5. Plot the points colored accordingly to their uncertainty.

  cm = plt.cm.get_cmap('RdYlBu')
  sc = plt.scatter(points[:,0], points[:,1], c=entropies, cmap=cm)
  plt.colorbar(sc)

  plt.xlabel('z mid phys')
  plt.ylabel('fault strike')
  plt.title("Covariance Matrix for {}{}".format(index[i],degree))
  plt.savefig('/content/drive/MyDrive/Final_project_ADS/Covariance_Matrix_
for_{}{}.png'.format(index[i],degree))

  i=i+1
```

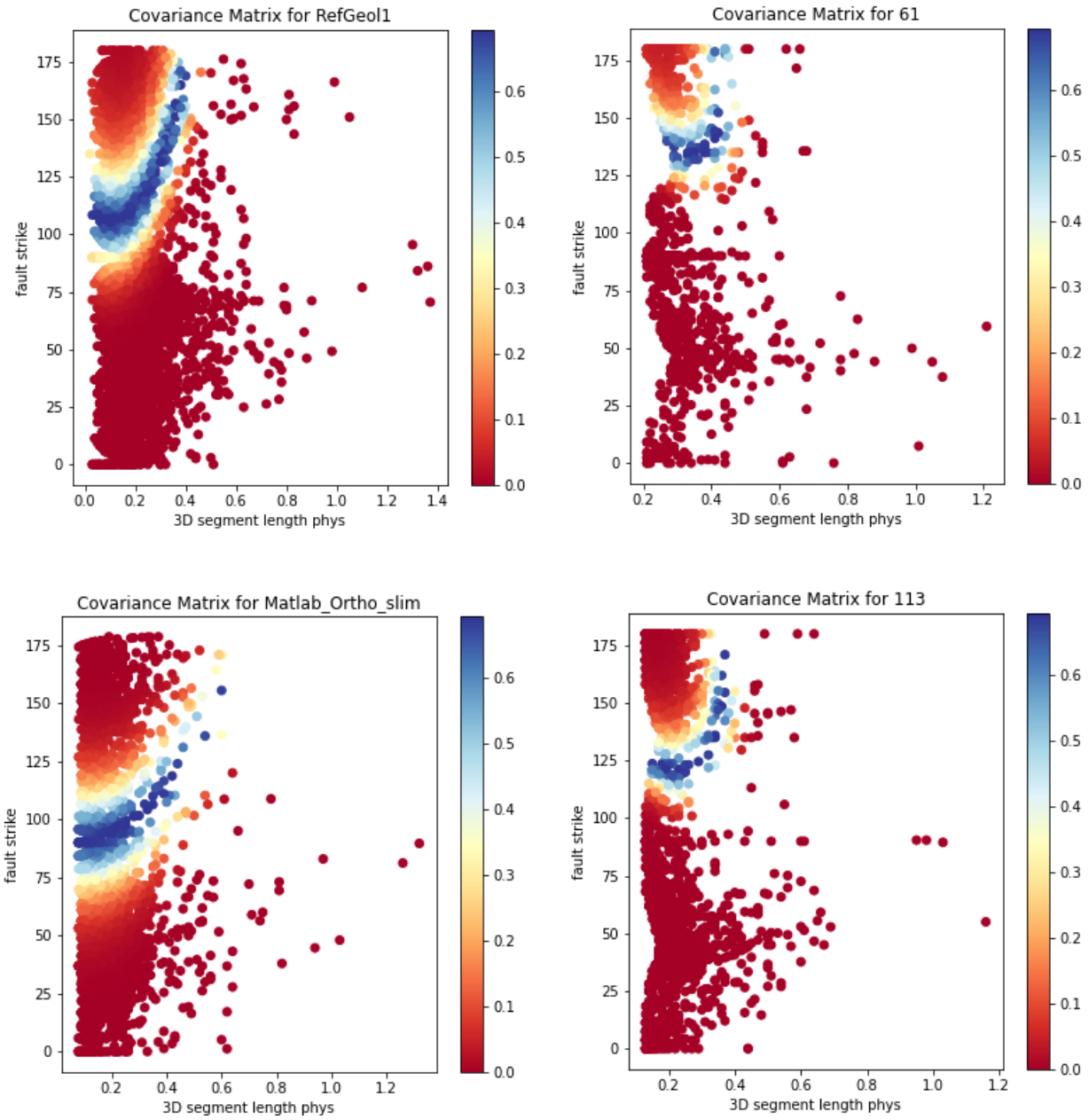

Fig. 38: Covariance matrix for 2 clusters, when plotting the 3D segment length against the fault strike.

4 Metadata

Each lineament set was produced with a specific set of parameters which were fed to the algorithm. These parameters (e.g. Illumination azimuth of the DEM, illumination angle/inclination of the light source for the DEM, resolution of the DEM & FIPH/cell size, coordinate system, file size, file format) need to be known for data reproduction, and hence are crucial metadata information. The metadata information needs to be stored in separate spreadsheet, listing all dataset used for further analysis in the framework of this project.

It seems reasonable to store the metadata with the code for data evaluation in github. The original DEMs are accessible via swisstopo in low resolution for large areas (federal office of topography). The DEM used in this work is not publicly accessible and hosted by Sandro Truttmann.

5 Data Quality

There are no specific quality requirements to be met, since we produce ourselves the dataset. However, the resolution of the DEM and the FIPH limit the precision of how detailed lineaments can be picked. The high resolution does not always provide the best results. Resolution requirements are mostly dependent on the scientific question one wants to address.

6 Data Flow

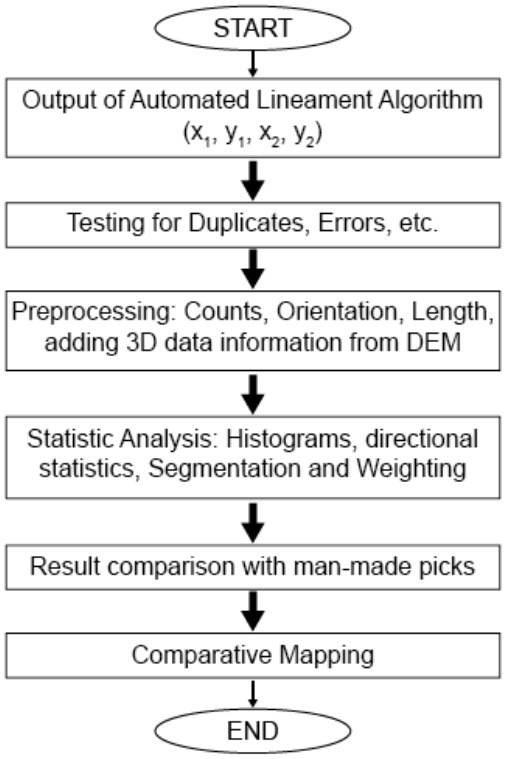

Fig. 39: Overview of dataflow from lineaments (database) to comparative mapping.

7 Data Model

The conceptual data model uses a DEM or orthophoto as general input. The DEM can be illuminated at various azimuth and illumination angles, each of them creating a different data input. Hence, with n different illuminations and m lineaments per Hillshade, we obtain a database of n x m lineaments (Fig. 40). This will be the database for further statistical evaluation. For the FIPHs and orthophotos, there are no different illuminations to be used. The orthophoto can be pre-processed with various filter and imaging techniques. Depending on the quality of the photo, multiple pre-processing steps are required and improve the performance of the detection.

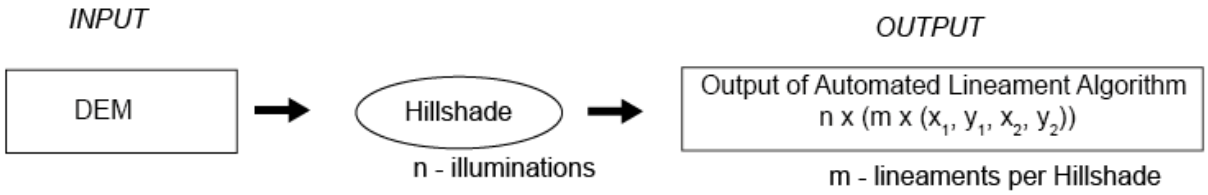

Fig. 40: Conceptual data model for a DEM.

Physical infrastructure needs: PC/ laptop with enough storage and memory (see details above) if performed locally (e.g. matlab-based algorithm). We used *Google Colab* for further data analysis, which requires only a computer with internet access.

8 Risks

There are certain risks while creating the database with matlab. Licenses run under academic license program that could expire. Hence, translating the automated lineament detection algorithm to an open source software (e.g. python), seems a good counter measure.

The computation time in matlab is in the order of a few hours to produce the database. Some testing is required, to find ideal setting parameters for a meaningful database creation. This is an iterative process and some basic data evaluation has to be performed to determine optimization potential.

Plotting and clustering in python was fast and takes usually no more than a few minutes. More time consuming models (e.g. CNN models) can be run under GPU for accelerated performance.

9 Conclusions

Overall, our data flow shows a way how lineament can be picked from digital elevation models and orthophotos in an automated fashion, that does not require prior knowledge or a trained geologist. Furthermore, the approach using an edge-detection algorithm with some defined settings removes a lot of subjectivity introduced by man-made lineament picking, and automates an otherwise time-consuming process.

In the following, the initially set goals are shortly answered:

a) Input data (multi/mono-directional hillshades, orthophotos) were plotted successfully, using various image filtering (e.g. Gaussian) and enhancement techniques (contrast, blurring etc.).

b) A python routine has been defined and various approaches for edge-detection have been tested. Generally, the dataset 113 (based on a multi-directional hillshade) seems to perform better for lineament detection than dataset 61 (orthophoto-based). Among different edge-detection algorithms (filter convolution via CNN model, Sobel, Canny-edge detection with Guassian blur), the Canny-edge detection algorithm achieved the best results for our specific purpose. We tried to optimize the settings for the edge detection and the Hough Transform using the Jaccard Score as performance indicator, combined with visual inspections and other statistical values.

c) The created lineaments were plotted and illustrated in various fashions, superimposed on hillshade or orthophotos. We tried to eliminate erroneous data and compared the datasets with the reference data compiled by a Geologist. However, more data cleaning would be required to fully account for wrong detections along the image frame (in both cases: matlab and python based routine).

d) We used various unsupervised machine learning algorithm to cluster the dataset. Additionally, we calculated the Silhouette Score to evaluate the optimal clustering number for different plots. Uncertainties for clustering were calculated via Covariance Matrix based on Gaussian Mixture fitting.

Overall, there is still a discrepancy between automated picked lineaments and manually picked lineaments from an expert. The matlab-based dataset seems to perform currently better than the python-based dataset. This is likely related to the use of Tensor Voting as additional step after edge detection, and before Hough Transforming the dataset. Therefore, in the future, we will introduce Tensor Voting in python to obtain better detection results.

10 Outlook

Supervised-machine learning was not used in this thesis, since an appropriate dataset for training did not exist. Generally, datasets with lineaments from geologist are rather sparse, why we focused on an approach that does not require any training dataset. Also, training a model using a reference dataset as in our case would lead to a trained model that is very site specific. We wanted to avoid that.

However, it would be still interesting to test the performance of a U-Net, using our reference dataset. This could be a project for the future, comparing a model output with the output of an unsupervised approach and test how the trained model performance at other test sites.

Acknowledgements

I used the infrastructure of the institute of Geological Sciences at University of Bern and appreciate it.

References and Bibliography

Burbank, D. W., and Anderson, R. S., (2011), Introduction to Tectonic Geomorphology. *Tectonic Geomorphology*, p. 1-16.

Linton, T., 2008, Tensor Voting Framework.

Rahnama, M., and Gloaguen, R., (2014a) TecLines: A MATLAB-Based Toolbox for Tectonic Lineament Analysis from Satellite Images and DEMs, Part 1: Line Segment Detection and Extraction. *Remote Sensing*, 6(7), p. 5938-5958.

Rahnama, M., and Gloaguen, R., (2014b) TecLines: A MATLAB-Based Toolbox for Tectonic Lineament Analysis from Satellite Images and DEMs, Part 2: Line Segments Linking and Merging. *Remote Sensing*, 6(11), p. 11468-11493.

Smith, M., and Pain, C., (2009) Applications of remote sensing in geomorphology. *Progress in Physical Geography*, 33(4), p. 568-582.

Wiemer, S., Giardini, D., Fah, D., Deichmann, N., and Sellami, S., (2009) Probabilistic seismic hazard assessment of Switzerland: best estimates and uncertainties. *Journal of Seismology*, 13(4), p. 449-478.

Le-Bourget-du-Lac (France), 16.08.2022

Stefano Fabbri
Liebefeld
Switzerland
stefano.fabbri@geo.unibe.ch
Wabersackerstrasse 23
3097