Department of Mathematics
and Statistics

MTHE 493 Thesis

Project B-4
Automated Core Logging

Charlie Horn
Mert Mumtaz
Liam Vovk
Mark Tamming

Supervisor: Dr. P. James McLellan

Submitted on: October 8, 2019

Abstract

Core logging is the systematic recording and measurement of geological information obtained from a piece of cylindrical rock. In particular, the successful computation of alpha and beta angles greatly aid geologists in assessing underlying rock masses and estimating deposit sizes of a given area. Currently, the process of measuring, recording, and analyzing this data is completed manually in a time-inefficient and error prone fashion. Mining companies must cover the cost of time lost logging core, and the costs associated with poor measurement of geotechnical angles which can have serious structural consequences.

This paper outlines an automated solution to this problem using images of drill core and custom computer vision software. After an extensive design process, the final solution incorporates Ellipse and Line Segment Detection for feature detection, Orthogonal Distance Regression for ellipse fitting , and trigonometric angle extraction for alpha and beta computations.

To test the efficacy of the proposed techniques, a case study was run using an actual image of a core sample. The proposed techniques detected 83% of the breakages and was capable of measuring alpha to +/- 5.6°, and beta to 8.6°. To improve the system, user input override, image pre-filtering, and advanced techniques to avoid converging on local minimum ellipse fits should be implemented.

Once active, the system will aid in smart mining decision-making. This ultimately reduces the social and environmental disruption mining often causes, and the economic costs associated with such unfortunate disruptions. Furthermore, it allows mining companies to more accurately identify and efficiently extract deposits, which increases profit.

1 Signatures

Name	Signature	Date
Charlie Horn		October 8, 2019
Mert Mumtaz		October 8, 2019
Liam Vovk		October 8, 2019
Mark Tamming		October 8, 2019
Dr. P. James McLellan		October 8, 2019

Contents

List of Figures

List of Tables

2 Introduction

Core logging is the systematic measuring and recording of geological information obtained from a piece of cylindrical rock drilled and removed from a mineral deposit. The extracted information is used to infer geological composition, structure, and behavior of the underlying rock mass. Such inferences relate to the lithology, mineralogy, structure, alteration zones and the geological history of the area. A major part of core logging pertains to the measurement of alpha and beta angles. These are the two geometrical properties of a joint, which is a naturally occurring fracture in the rock masses. Once measured, these angles are essential in helping geologists better understand the characteristics of a land mass. These characteristics are most commonly used for the construction of a mine.

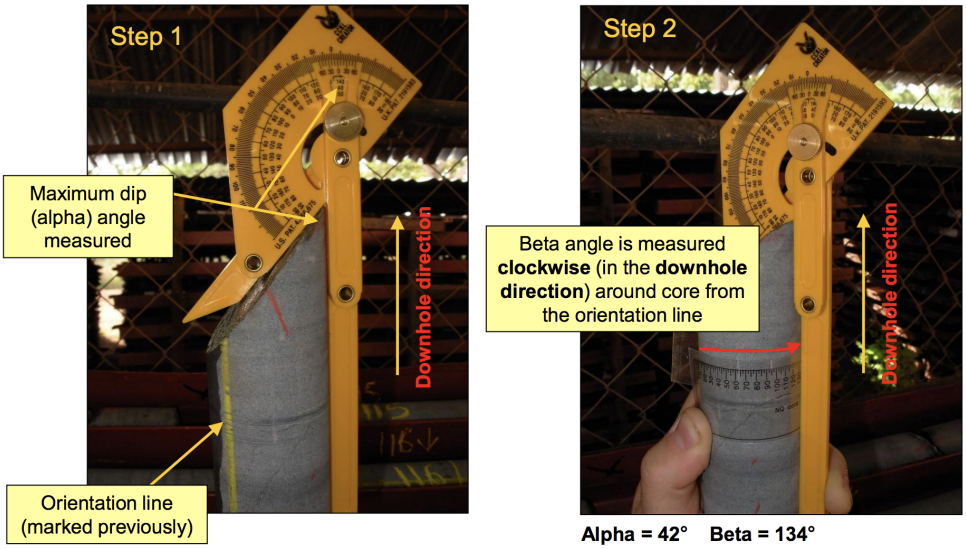


Figure 1: Measurement of alpha and beta in industry [1]

.

Currently, the process of measuring alpha and beta angles is an entirely manual process, as seen in Figure 1. The taking of numerous precise and accurate measurements of core samples consecutively can be very tedious, time-inefficient, and susceptible to human-error. Since this information is then commonly used by engineers for mining and geological projects, it is essential that these measurements be precise. The need for automation is apparent.

3 Problem Description

Core logging is considered the primary method for determining the grade, size, profit of a mineral deposit and most importantly; the structural stability of a given land mass. A geotechnical model that describes rock core is typically given by the combination of manual and automatic procedures for obtaining geotechnical measurements. Technological advancements are attempting to automate core logging. However what is currently available in the market is cost-prohibitive and can also be too cumbersome for mining companies to transport. The current industry standard for

creating these geotechnical models is to hire a junior geologist, and pay their salary and travel costs.

In addition to wages, mining companies pay an implicit price by making decisions off of inaccurate alpha and beta measurements, which can be seen in Figure 2. By current standards, rock mass assessments are often erroneous and can have large error margins, implying that a solution adds value by providing both time savings and accuracy.

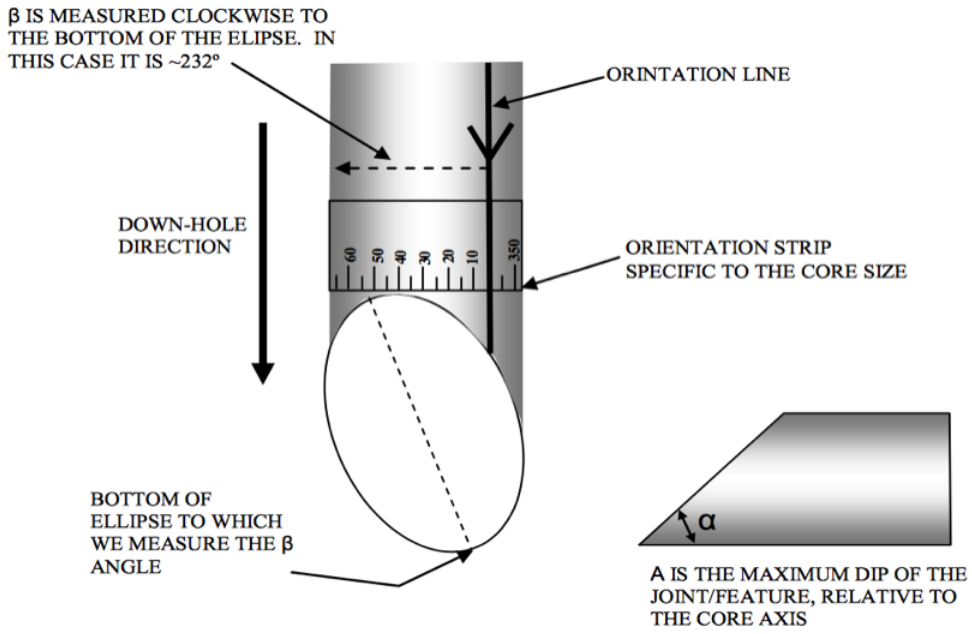


Figure 2: Illustration of alpha and beta angles [1]

Currently, a geologist measures the alpha angle by using a carpenter's angle to measure the angle between the steepest incline of the fracture relative to the vertical axis of the core. The beta angle is measured by placing a calibrated strip at the orientation line and measuring in a clockwise direction up to the top of the ellipse formed by the fracture (note that this is 180°from the bottom of the ellipse). The beta of a joint indicates the direction that the rock could break away or slip. The alpha angle of a joint is important because it indicates the steepness of a joint.
These measurements allow geologists to decide where to look for more minerals, and erroneous calculation can cause them to completely miss or overestimate the size of deposits. Using industry standards, the total cost of these geotechnical measurements can be quantified in the annual expense of junior geologists, the opportunity cost of missing or over estimating the size of a mineral deposit, and the social and environmental costs of implementing mining infrastructure on unsafe ground. The goal of this project is to produce a portable, automated, computer vision system that can calculate the alpha and beta angles of fault lines, while reducing operating costs and the potential for human error.

There is very limited literature aimed at remedying this inefficiency. The only known attempts at this problem from an industrial perspective are a spectral imaging solution provided by Specim Ltd. and an x-ray diffraction imaging technique provided by Bruker [11][12]. The former utilizes imaging spectrometers in order to visualize mineralogical and structural information while the latter analyses the scattered intensity of an x-ray beam hitting certain geological objects. While effective,

these solutions are both very specialized and heavily utilize chemical processing methods which can be unnecessary and very cost-inefficient for most general purposes. With regards to academic work, perhaps the most well-known method of automated extraction of geological features is to use Light Detection and Ranging (LiDAR) [13]. Although there are definite improvements in the ability to characterize and extract measurement at large scales with very reliable results with this approach, it has immense drawbacks in terms of feasibility. Not only is it very expensive in terms of cost and space, it possesses safety implications for the use of high powered laser equipment, while lacking practicality and robustness in varied field conditions [14]. Kissi, proposes a Structure from Motion (SfM) reconstruction of geological sites using UAV drones in order to automate certain geological measurements [14]. While sufficient for its intended purpose, this approach requires the extra challenges of UAV Drone programming and dynamics, very expensive computational power, and is in very early stages for any serious implementation.

Due to the current inadequacies of the outlined process as well as the lack of suitable solutions, the need for a different and improved approach is apparent. A reliable automated approach is proposed. This new approach will accurately identify and extract alpha and beta angles from drill cores in a timely fashion while remaining cost-effective in an effort to differentiate from the current attempted solutions in place today.

4 Mathematical Background

4.1 Ellipse and Line Segment Detector

4.1.1 Feature Detection

The automatic detection and extraction of elementary geometric features such as line segments and elliptic curves in images dates back to the birth of computer vision itself [15]. Although taken for granted in modern technologies, the task itself is far from trivial and researchers have approached it in various ways in the past. For purposes which will be made clear in the Design Process, the Ellipse and Line Segment Detector (ELSD) will be used, which is an extension of the original Line Segment Detector (LSD) developed by Randall et al. [3]. The ELSD is composed of 3 major steps: region growing and hypothesis selection, *a contrario* validation, and finally, model selection. What follows is a detailed explanation of the underlying mathematical models of each stage [2].

4.1.2 Feature Hypothesis Selection

Hypothesis selection is the process of labelling certain regions in a given image as potential geometrical shapes, i.e., line-segments or ellipses. The main feature of the ELSD is that this process requires no tuning. The method consists of two steps: region growing for line-segments, and curve growing for ellipse segments. The former is achieved primarily with the LSD while the latter utilizes the appropriately named ELSD [3].

Region growing starts with computing the gradient of one pixel, referred to as the seed pixel.The gradients at this pixel are given by:

$$g_x(x,y) = \frac{i(x+1,y) + i(x+1,y+1) - i(x,y) - i(x,y+1)}{2},$$

$$g_y(x,y) = \frac{i(x,y+1) + i(x+1,y+1) - i(x,y) - i(x+1,y)}{2},$$

This is then computed successively for each pixel in the image. Gradients are computed in order to produce a level-line vector field. This field is then segmented into connected regions of pixels with the similar level-line orientation up to a certain tolerance, denoted by τ. These are referred to as the line support regions and each region is then considered a candidate for a line segment (to be validated in step 2). See Figure 3.

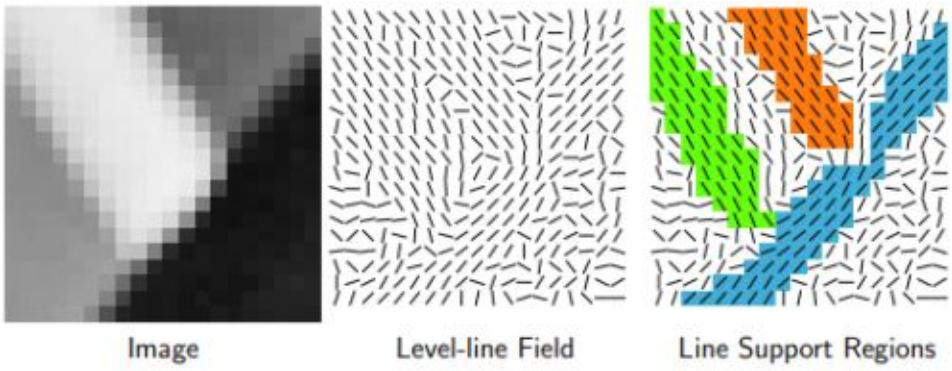


Figure 3: Illustration of level-line fields and support regions [2]

Note that each distinct colored rectangle is a potential hypothesis and will be subject to the validation procedure, described in the next section. The pixels in the rectangle whose level-line angle corresponds to the angle of the rectangle up to the tolerance, τ, are called *aligned points* as observed in Figure 4. This implies by definition that the difference in angle of two level lines cannot differ by more than 2τ.

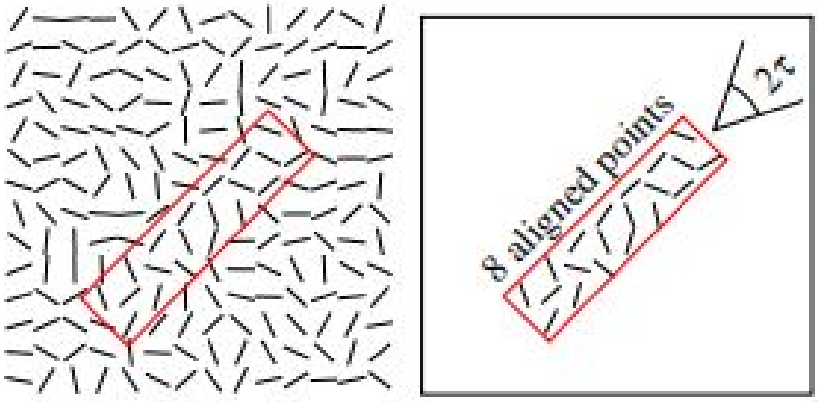


Figure 4: Illustration of the tolerance angle [3]

The total number of pixels in the rectangle/region, n, and its number of aligned points, k, are counted and used to validate, or not validate, the rectangle as a detected line segment. In essence, once all the rectangular regions in the given area are computed, the region growing section for line-segments is concluded. To recap, starting from a seed pixel, a region growing process recursively

groups pixels into connected regions sharing the same gradient orientation i.e., level-line angle, up to a given precision tolerance, τ, until all pixels are visited.

For elliptical arcs (which includes circular arcs), the ELSD proposes a curve growing procedure that adds a second level of procedural groupings, which builds on top of the region growing process proposed by the LSD discussed above. As opposed to chaining together only those regions which share similar orientation, two additional constraints are imposed:

1. **Convexity**: impose that consecutive rectangular pairs turn in the same direction so that $\Delta\Theta_i$ and $\Delta\Theta_{i+1}$ have the same sign.
2. **Smoothness**: roughly imposed by restricting chaining to those regions whose level-line orientation differ by less than $\frac{\pi}{2}$.

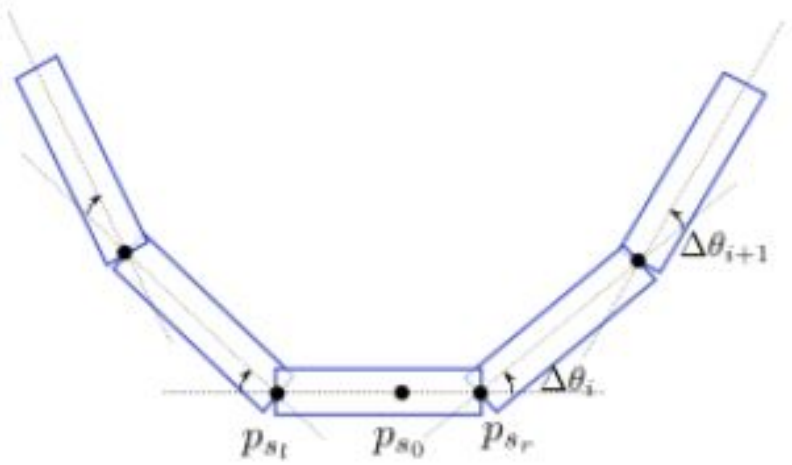


Figure 5: Curve Growing [2]

A visual illustration of these constraints is found in Figure 5. Similar to the region growing case, curve growing produced a polygonal approximation of a curve by employing a recursive scheme. Once the number of regions to be added reaches its limit (i.e., no more pixels left), the five parameters of an ellipse are computed, fitting the outlined pixels.The exact techniques for this fitting procedure are similar to the procedure proposed by Rosin, the difference being the additional computation of delimiting angles and ring widths, which are precisely discussed at length in [2][16] .

Once curve growing and region growing for a given image are finished, a myriad of potential feature hypotheses are produced; line-segment candidates, and ellipse candidates. Before being deemed as definite features, they go through a validation procedure.

4.1.3 Validation of Detected Features

The main idea for validating features in the ELSD comes via the *a contrario* approach and the Helmholtz principle proposed by Desolneux et al. [4] . The principle essentially states that no detection should be produced on an image of noise. Correspondingly, the *a contrario* approach proposes to define a noise or *a contrario* model, H_0, where the desired geometrical feature is not

present. It follows that a detection is then deemed valid if the expected number of detections as good as the observed one in the image is small on the noise model. In essence, structured and reasonable detections should be defined as being very rare in H_0. As with the hypothesis selection step, this procedure can be split into the line-segment validation and the ellipse validation.

In the context of line segment regions, recall that the pixels in a region whose level-line angles differ by at most a given tolerance, τ, are called *aligned points* denoted by k, as illustrated previously in Figure 3 . Then, in following the *a contrario* approach, the event that a line segment in the noise model, H_0, has at least the same number of aligned points, k, as in the original observed image is of particular interest. Given some image i, and a region/rectangle, r, the number of aligned points is denoted $k(r, i)$ and the total number of pixels in the region is denoted $n(r)$. It follows then that the number of detections or events, which are at least as good as the observed noise-free case is given by:

$$N_{test} P_{H_0}[k(r, I) > k(r, i)] \tag{4.1.1}$$

where N_{test} denotes the total number of possible regions/rectangles considered, P_{H_0} denotes the probability on the model H_0 (which will be defined shortly), and I is a randomly produced image in accordance with H_0. It should be emphasized that the H_0 stochastic model fixes the distribution of the aligned points, $k(r, I)$, which is clearly only dependent on the level-line field produced for I. In essence then, H_0 is solely a noise model for the gradient orientation as opposed to the actual image in a macroscopic context. Now, the *a contrario* model H_0 is defined as a stochastic model of the level-line field satisfying the following conditions [2] :

1. $\{LLA(j)\}_{j \in Pixels}$ is composed of random variables independent of each other.
2. $LLA(j)$ follows a uniform distribution within the interval $[0, 2\pi]$.

where $LLA(j)$ is the level-line angle of a pixel j.It has been proven by Desolneux et al. that these two assertions hold under certain conditions of subsampling if I is a Gaussian white noise image [2] . Under the model H_0 then, the probability that an arbitrary pixel is an aligned point is:

$$p = \frac{\tau}{\pi}$$

Moreover, due to the independence of the random variables $LLA(j)$, $k(r, I)$ will naturally give rise to a binomial distribution. Finally, the aforementioned probability on the noise model is given by [4] :

$$P_{H_0}[k(r, I) > k(r, i)] = B\Big(n(r), k(r, i), p\Big) \tag{4.1.2}$$

where $B\Big(n(r), k(r, i), p\Big)$ is the well-known binomial distribution tail as given by:

$$B(n, k, p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

Recall, N_{test}, the total number of possible regions/rectangles that could show an alignment. Before computing this value, it is important to note that all regions are directed. For example, a rectangle starting at point A and ending at point B is not considered the same as its converse. So in an NXM image then, $NMxNM$ different rectangles must be considered at the very least. Also $NM^{\frac{1}{2}}$ different widths of a rectangle need to be added in, see Figure 6.

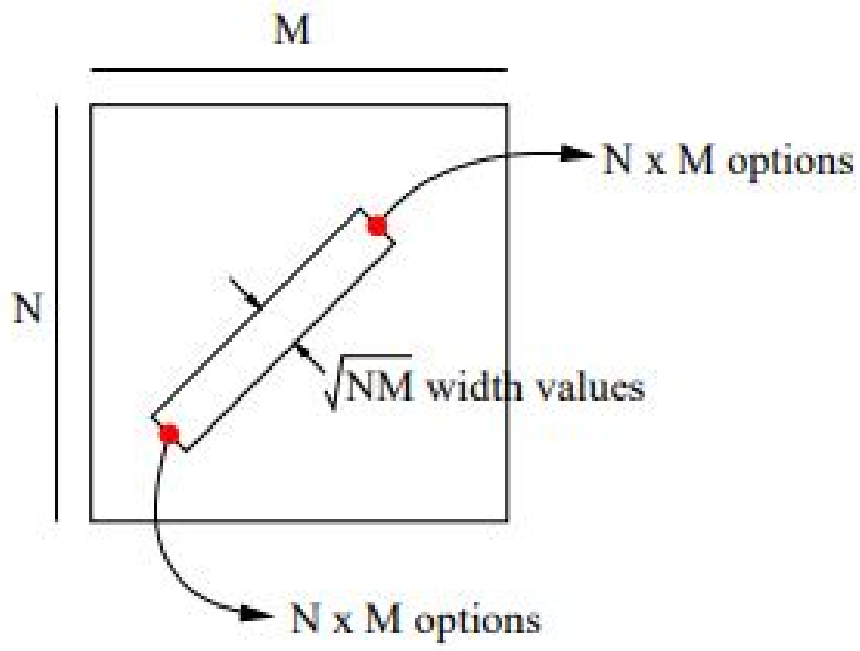


Figure 6: Number of different rectangles to be considered [4]

Therefore the total number of rectangles considered is finalized as:

$$(NM)^{\frac{5}{2}}$$

The final piece in the set up of the validation process for a line-segment is that of the Number of False Alarms (NFA) for a given region and image, defined by:

$$NFA(r,i) = (NM)^{\frac{5}{2}} B\Big(n(r), k(r,i), p\Big)$$

This expression defines the expected number of regions/rectangles which have an adequate number of aligned points, k, to be as rare as r under H_0. Informally, when the NFA is large, this correspond the event/feature being expected on the *a contrario* model, which means the feature is common and thus probably irrelevant. In contrast, a small NFA means that the event is rare and meaningful. In essence, this concludes the validation set-up for the line-segment section. Once the NFA is minimized, the detected feature is deemed valid. This will be formally shown later, for now, the ellipse validation still remains.

Recall that a set of pixels are said to be aligned if their level-line angles fell within a tolerance, τ For ellipse segment detection however, this definition needs to be restructured. A pixel, p, is said to be aligned with a circular or elliptical arc, a , up to a tolerance, σ if:

$$Angle(\nabla p, dir_{\perp}(tan_a(p))) \leqslant \sigma\pi$$

For an intuitive understanding of this concept, refer to Figure 7.

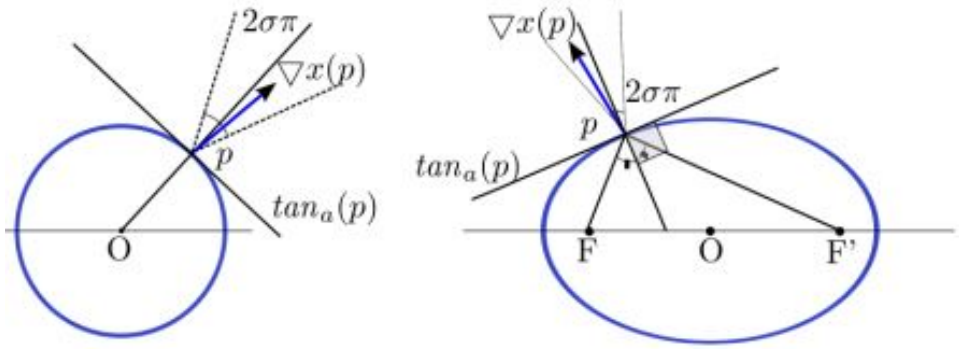


Figure 7: Alignment of circular and rectangular candidate pixels [2]

Once the aligned pixels are established, the noise model needs to once again be defined. The *a contrario* model for line segment detection, actually seamlessly can be applied to detection of ellipses as well [16]. Recall once again that H_0 isn't necessarily a model of real noise in images but rather a model for unstructured, isotropic zones of the image where aligned structures aren't perceived. There is a thorough discussion on the effectiveness of this method and the results were as impressive as human perception [16] .

The only actual remaining step for the definition of NFA for the ellipse then is the number of possible different ellipse configurations, denoted by, N_{test} as with the line-segment case. Note that elliptical arcs have eight degrees of freedom: 2 centres, 2 axes, 1 orientation, 1 width, and 2 delimiting angles, as seen in Figure 8.

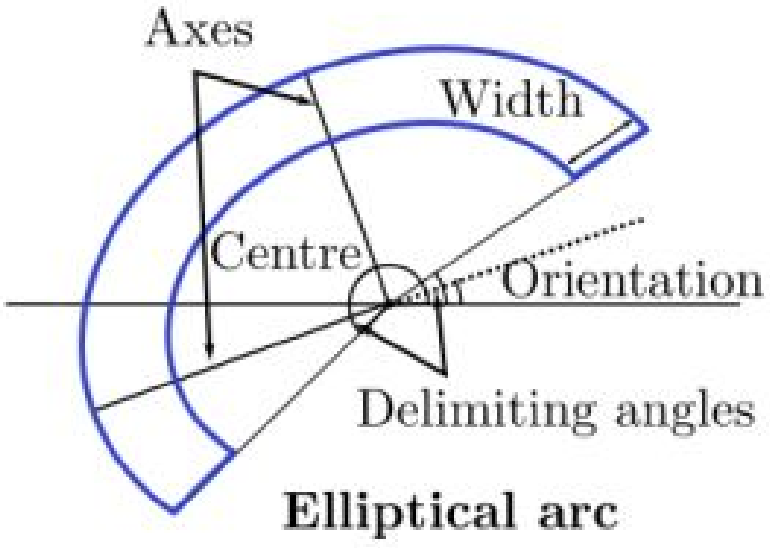


Figure 8: Degrees of freedom for an ellipse [4]

It follows then, that the different possible ellipses considered for an MxN image, is given by MN^4. Finally, the NFA for the ellipse is then defined as:

$$NFA(r,i) = (NM)^4 B\Big(n(r), k(r,i), p\Big)$$

where the only difference between the ellipse and line detection is due to different alignment metrics and the number of possible geometric shapes considered. As a final note on validation, note the following result proposed (and proved) by Desolneux et al [16]:

$$E_{H_0}[\sum_r NFA(r,I) < \epsilon] \leqslant \epsilon$$

This result says the average number of $(\epsilon - meaningful)$ under the H_0 model is less than ϵ. In other words, the number of detection on the noise model is controlled with ϵ and it can be modified as desired. However, Moisan, Morel, and Desolneux found that $\epsilon = 1$ is optimal for line and ellipse segment detection [16]. Note that this means the approach assumes the risk of one false detection on average per image.

4.1.4 Model Selection

The hypothesis or candidates declared $(\epsilon - meaningful)$ in the validation phase are subsequently placed in a model selection phase for the best interpretation of the resulting data and the winner is deemed as a final valid detection. The model selection theory is a central subject in statistical learning and an important number of model selection criteria is available in the literature [16]. The ELSD entails a model selection step within a linear regression problem: given a set of pixels, and two fitted candidates (line,ellipse), determine which model is the most suitable to explain the data. The NFA is utilized as a model selection criterion, i.e., the candidate possessing the smallest NFA is considered as most meaningful, and kept as final valid detection. This usage is actually due to numerous lengthy and thorough discussions also available in literature [16]. For brevity, it is pointed out qualitatively that the NFA follows the Occams razor principle. In particular, it contains a term illustrating the goodness of fit (the binomial tail) and a term penalising for complexity (the number of candidates N_{test}, proportional with the number of free parameters).

4.1.5 Feature Detection Pseudocode

Once the intuition for the a contrario model and its role in the derivation of NFA is developed, the implementation of this method becomes apparent as seen below in Figure 9.

Input: Gray-level image x, **parameters**: none
Output: $\mathcal{L}_f$ – list of valid features (line segments, circular arcs, elliptical arcs).

```
grad ← compute_gradient(x);
foreach pixel p_i in x do
    R ← region_grow(p_i, grad);
    C ← curve_grow(R, grad);
    line ← fit_rectangle(R);
    circle ← fit_circular_ring(C);
    ellipse ← fit_elliptical_ring(C);
    (NFA_line, NFA_circle, NFA_ellipse) = NFA(line, circle, ellipse);
    NFA_min ← min(NFA_line, NFA_circle, NFA_ellipse);
    if NFA_min ≤ 1 then
        add feature corresponding to NFA_min in L_f;
    end
end
```

Figure 9: ELSD Pseudocode [4]

The input image, assumed to be gray-scale, is looped through pixel by pixel. The hypothesis selection steps of region grow (group pixels with similar gradient orientation) and curve grow (region grow with added constraints of smoothness and convexity) are applied at each pixel. Once this is done, candidate regions are outlined (see rightmost image in Figure X) into rectangular or elliptical regions. Then, the NFAs for each geometric candidate are computed, and the most likely shape, i.e., the shape with the least expected number of false alarms is deemed chosen as the most likely interpretation

4.2 Clustering

4.2.1 Clustering Description

The K-Means clustering algorithm aims to separate N discrete data points $\mathcal{X} = \{x_1, x_2, ..., x_N\}$ into K spatially distinct partitions $\mathcal{S} = \{S_1, S_2, , S_K\}$. These partitions aim to optimize a distance measure which is based on data attributes, which will be taken as (x, y) pixel coordinates. This is done by selecting K random data points as initial centroids $\mathbf{c}_0 = \{c_1, c_2, ..., c_K\}$, and follows the iterative two step process outlined below.

1. The nearest neighbour condition simply maps each data point in $\mathcal{X}$ to its closest centroid in $\mathbf{c}_t$.

$$S_i = \{x \in X \mid |x - c_i| \leq |x - c_j| \; \forall \; j \in (1, \; 2, \; ..., \; N)\} \tag{4.2.1}$$

2. An updated centroid set is then calculated as the center of each partition.

$$c_i = \frac{1}{|S_i|} \sum_{1}^{K} x_i \tag{4.2.2}$$

The partitions are then updated, and this process is iteratively carried out until the centroids have converged. This means that Expression 4.2.3 holds at time t.

$$D \;=\; d(c_t, S_t) \;-\; d(c_{t-1}, S_{t-1}) \;\leq \epsilon \tag{4.2.3}$$

Where $d(c_t, S_t)$ is a distant metric defined as:

$$d(c_t, S_t) \;=\; \sum_{i=1}^{K} \sum_{S_i} |x_j \;-\; c_i| \tag{4.2.4}$$

4.2.2 Clustering Pseudocode

The K-means clustering algorithm has a low level of complexity. It is an iterative method that has a pseudocode shown in 10.

{Given $X = (x_1, \cdots, x_n) \in \mathbb{R}^{m \times n}$, partition it into K clusters $S_1, \ldots, S_K$.}
Initialize K prototypes $p_1, \ldots, p_K$, randomly or by another clustering algorithm.
repeat
  Set $S_j := \emptyset$ for $j = 1, \ldots, K$.
  for $i = 1, 2 \ldots, n$ **do**
    Find k such that $\|x_i - p_k\| \leq \|x_i - p_j\|$ for $j = 1, \ldots, K$.
    Set $S_k := S_k \cup \{x_i\}$.
  end for
  for $j = 1, 2, \ldots, K$ **do**
    Set p_j = the mean of points in S_j.
  end for
until it converges (i.e., $p_1, \ldots, p_K$ unchanged).

Figure 10: K-Means Clustering Pseudocode [5]

4.3 Orthogonal Distance Regression (ODR)

4.3.1 ODR Description

The derivation of the orthogonal distance regression algorithm (ODR) used in this project is a result of the research of Boggs and Rogers [17]. The following implementation is used to find the maximum likelihood estimators of parameters in non-linear errors-in-variables models (models in which there is significant error in the independent variables [17]).

Proceed by defining the observed random variables to be,

$$X_i \qquad i = 1, 2, ...n,$$

with underlying true values given by

$$x_i \qquad i = 1, 2, ...n,$$

where $x_i \in \mathbb{R}^m$, $m \in \mathbb{Z}_+$. Next it is assumed that there is an error δ_i associated with each x_i. In

this only implicit relationships are dealt with, meaning that there are no distinguished dependent variables. Thus the observed variables to be fit, given by $\boldsymbol{\beta} \in \mathbb{R}^p$, $p \in \mathbb{Z}_+$, must satisfy the following equation:

$$f(X_i + \delta_i; \boldsymbol{\beta}) = 0 \qquad i = 1, 2, ...n.$$

The function f must be either a linear or non-linear smooth function of these arguments. Note that throughout this paper, bold face is used to denote the matrix or column vector whose components are the corresponding subscript values, e.g., $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_p)^T$, it is also assumed that X_i is one dimensional to simplify notation.

The procedure for estimating $\boldsymbol{\beta}$ in the errors-in-variables model must account for X_i's error, δ_i. This is done by defining an orthogonal distance, r_i, from the point X_i to the curve $f(\tilde{x}; \tilde{\boldsymbol{\beta}})$. Here $\tilde{}$ represents any arbitrary value of the variable. Under the assumption that $\delta_i \sim \mathcal{N}(0, {\sigma_{\delta_i}}^2)$, the maximum likelihood estimate $\hat{\boldsymbol{\beta}}$ minimizes the orthogonal distance r_i. Therefore it must be a solution to the equation for r_i, that is:

$$\begin{aligned} {r_i}^2 = \min_{\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\delta}}} & \sum_{i=1}^{n} \left\{ w_{\delta_i} {\delta_i}^2 \right\} \\ & \text{subject to: } f(X_i + \delta_i; \boldsymbol{\beta}) = 0 \quad i = 1, 2, ...n. \end{aligned}$$

where $w_{\delta_i} > 0$ are weights that can be used to eliminate observations from the analysis or to modify the effect of δ_i on the fit [18]. By linearizing the constraints as previously done by Gulliksson and Soderkvist, one can obtain an equivalent unconstrained minimization problem that will be defined as the *Orthogonal Distance Regression Problem* (ODR) [19]. This is given by

$$\min_{\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\delta}}} \sum_{i=1}^{n} \left\{ [f(X_i + \delta_i; \boldsymbol{\beta})]^2 + {w_{\delta_i}}^2 {\delta_i}^2 \right\}. \tag{4.3.1}$$

There are several methods of solving the ODR problem, in the following, the methods given by Boggs, Byrd, and Schabel, as well as Gulliksson and Soderkvist are reviewed. [19] [20]. In this method $w_{\delta_i} = 1$. Under these conditions ODR is shown to be superior to OLS. The algorithm is a construction of the *trust-region* strategy that results in a Levenberg-Marquardt type step.

It can be shown that the equation (4.3.1) can be viewed as an extended ordinary least squares (OLS) problem [19] [20]. Let:

$$g_i(\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\delta}}) = \begin{cases} f(X_i + \tilde{\delta_i}; \tilde{\boldsymbol{\beta}}), & i = 1, ...n \\ w_{\delta_i} \tilde{\delta_i}, & i = n+1, ...2n. \end{cases}$$

Now (4.3.1) can be written as

$$\min_{\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\delta}}} \left\| \boldsymbol{g}(\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\delta}}) \right\|_2$$

which is an OLS problem with $(n + p)$ parameters and 2n observations. For ease of notation, the following is defined

$$\tilde{\boldsymbol{\theta}} = \begin{bmatrix} \tilde{\boldsymbol{\beta}} \\ \tilde{\boldsymbol{\delta}} \end{bmatrix}$$

and the Jacobian of

$$\boldsymbol{g}(\tilde{\boldsymbol{\theta}}) = (g_1(\tilde{\boldsymbol{\theta}}), g_2(\tilde{\boldsymbol{\theta}})..., g_2 n(\tilde{\boldsymbol{\theta}}))$$

to be

$$J_{ij} = \frac{\delta g_i}{\delta \tilde{\theta}_i} \ , \qquad \boldsymbol{J} \in \mathbb{R}^{2n \times (n+p)}.$$

Then the first two terms of the Taylor series are written as

$$\boldsymbol{g}(\boldsymbol{\theta}^c) + \boldsymbol{J}(\boldsymbol{\theta}^c)\boldsymbol{s} \ \triangleq \ \boldsymbol{g}^c + \boldsymbol{J}^c\boldsymbol{s} \tag{4.3.2}$$

which is the *linearized model* at the current iteration, θ^c, where

$$\boldsymbol{s} = \tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}^c.$$

From (4.3.2) a step can be computed, $\boldsymbol{s}^c$, from which the next iterate, $\boldsymbol{\theta}^+ = \boldsymbol{\theta}^c + \boldsymbol{s}^c$, is found. The obvious step is the so called Gauss-Newton step where $\boldsymbol{s}^c$ is just the solution of

$$\min_{\boldsymbol{s}} \|\boldsymbol{g}^c + \boldsymbol{J}^c\boldsymbol{s}\|_2 \,,$$

however, Dennis and Schnabel showed that even when controlling the size of the step, e.g., by line search, the resulting algorithm has not proven to outperform other procedures [21]. If, however, the size of the step $\boldsymbol{s}^c$ is controlled by the extent to which (4.3.2) is a *good* model of $\boldsymbol{g}$, then a much better procedure arises [17].

In order to implement this, the *trust-region algorithm* was used, where the size of the trust region is directly proportional to how well (4.3.2) is a *good* model of $\boldsymbol{g}$. The algorithm chooses the next step $\boldsymbol{s}^c$ as the solution to

$$\begin{aligned} &\min_{\boldsymbol{s}} \|\boldsymbol{g}^c + \boldsymbol{J}^c\boldsymbol{s}\|_2 \\ &\text{subject to: } \|\boldsymbol{s}\| \leq \tau \end{aligned}$$

where τ is the *trust-region radius*, the region where (4.3.2) is a *good* model of $\boldsymbol{g}$. At each iteration the value of τ is adjusted based on whether $\|\boldsymbol{g}(\boldsymbol{\theta}^c + \boldsymbol{s}^c)\| < \|\boldsymbol{g}^c\|$ and on a comparison of $\|\boldsymbol{g}(\boldsymbol{\theta}^c + \boldsymbol{s}^c)\|$ with $\|\boldsymbol{g}^c + \boldsymbol{J}^c\boldsymbol{s}\|$. By exploiting the structure of the Jacobian matrix $\mathbf{J}$, the ODR problem as presented can be solved in $O(np^2)$ time, equivalent to the efficiency of standard solutions to the OLS problem [17]. Thus, using this algorithm, an ODR problem can be solved as efficiently as an OLS problem to find the maximum likelihood estimators of parameters in errors-in-variables models.

4.3.2 ODR Pseudocode

Next, a pseudocode implementation of the trust-region algorithm will be shown. Given a guess at the initial conditions, $\boldsymbol{\theta}^c$ and an estimate of the trust-region, τ:

1. Solve (4.3.2) for $\boldsymbol{s}^c$
2. Set $\boldsymbol{\theta}^+ := \boldsymbol{\theta}^c + \boldsymbol{s}^c$
3. Test: **If** $\|\boldsymbol{g}(\boldsymbol{\theta}^+)\| < \|\boldsymbol{g}^c\|$
   - set $\boldsymbol{\theta}^c := \boldsymbol{\theta}^+$
   - adjust τ if necessary
   - check convergence

   else
   - reduce τ
4. Repeat from Step 1, if no stopping conditions are met [17].

The iterations are stopped when any one of three stopping criterion are satisfied. These are the *sum of squares convergence* that indicates when the change in (4.3.2) is sufficiently small, *parameter convergence* which indicates that the change in estimated values of $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ is sufficiently small, and finally when the limit on number of iterations is reached [18].

5 Economic and Legal Implications on Design

Before beginning the design process, one must keep in mind the context of this project and the constraints that come with it.

5.1 Economic Implications on Design

Canada has one of the largest mining supply sectors globally with more than 3,700 companies supplying engineering, geotechnical, and environmental services to mining operations [22]. The industry accounts for 19% of the value of Canadian goods exports (in 2016) and yields a mineral production value of $40.8 billion, while contributing $57.6 billion to Canada's Gross Domestic Product (GDP) [23]. Moreover, it is estimated that roughly 70% of the world's mining companies are based in Canada [23]. In particular, factors such as changes in manufacturing techniques, and the development of high-performance drill heads and core bits have increased the feasibility of mining equipment, thereby propelling the industry growth. In fact, the global mining drills and breakers market alone is expected to reach USD 17.03 billion by 2025 [24]. In Quebec alone, core drilling costs accounted for $135,603,572 in 2016 and was the result of drilling 989,954 meters of core [25]. This is roughly a 16% increase from 2015 as well as a 20% increase from 2014, see Figure 11.

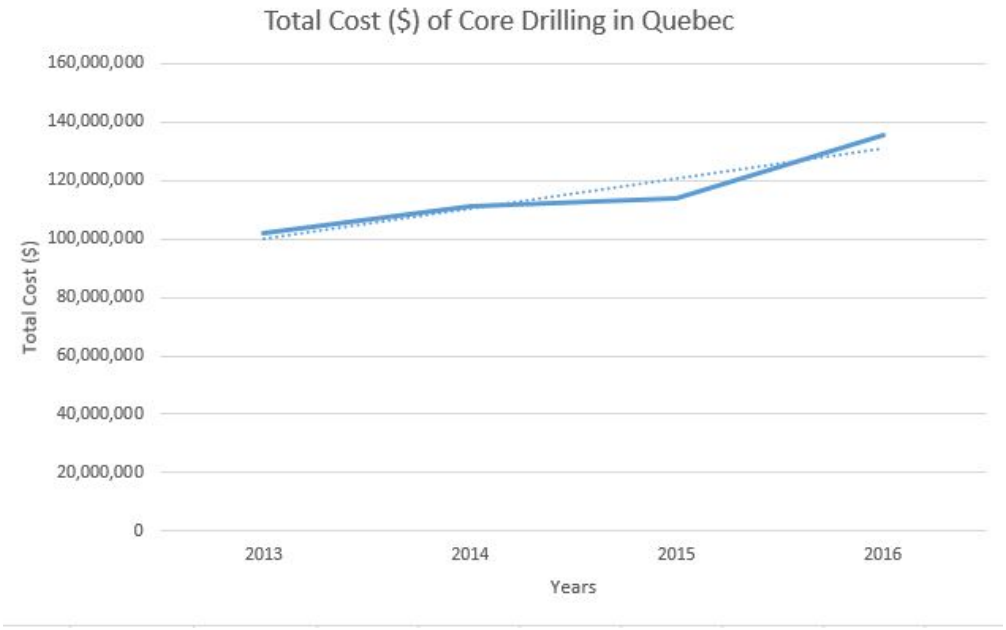


Figure 11: Drill Core Costs vs. Time

It is clear that this market is attractive for many engineering firms due to its size and growth rate. Industry competitiveness requires the best possible quality-price combination for any engineering product. As a result, any successful design will require economic insight and justification during the design stage in order to appropriately assess trade-offs among alternatives and competitors. At the time of this paper, there is only one direct competitor, Coreprofiler [26]. Other indirect competitors/alternatives exist but attempt to solve different problems in larger problem spaces than just drill core analysis. Thus they won't be considered. The size of the market is large and the growth rate is ever increasing, therefore a successful design needs to have higher performance and reduced cost when compared to the competition.

Coreprofiler is a software system developed by Datamine in order to perform automatic geotechnical analysis of given core images. Users have to take photos of core trays, input them into the software, place depth markers, add core quality information, set intervals, define log sheets, and scale certain images during the process. The company has 3 revenue models, offering a full license for $6,000, a monthly subscription for $750, and a yearly subscription for $4,000 [26]. The inefficiencies with this approach are similar to those of the manual process of core logging, namely the lack of any significant automation. For example, the current industry standard for logging and analysis of one core is about 35 minutes. Comparatively, the Coreprofiler claims to occupy anywhere from 3 to 17 minutes. It is apparent then, in order to differentiate from the competition, a successful design needs to minimize manual involvement while improving processing times. These also need to be achieved without sacrificing accuracy. If success is defined by capturing at least 50% of the current market with a business model that only operates in full licenses, then an expected yearly revenue of at least $11,100,000 can be generated. This is assuming full channel distribution, using a pricing scheme similar to competitors like Coreprofiler. The scheme is defined as follows:

$$(50\%\, market)(3,700\, Mining\, Companies)(100\%\, channel\, dist.)(\$6,000\, per\, license) = \$11,100,000$$

On top of generated revenue, a design satisfying the previously specified conditions will result in the following savings for mining companies:

$$L = \textit{Length of core drilled annually} = 1600000\ m\ [27]$$
$$p = \%\ \textit{of core requiring } \alpha \textit{ and } \beta \textit{ measurements} = \%10\ ^{1}$$
$$R = \textit{Rate of measurement} = 10\ \textit{minutes/meter}\ ^{1}$$
$$T_{Current} = \textit{Current annual time spent measuring} = R * L * p = 26700\ \textit{hours}$$
$$T_{Saved} = \textit{Time spent measuring with this technology} = T_{Current} * \frac{30\ \textit{seconds}}{35\ \textit{minutes}} = 254\textit{hours}$$
$$W = \textit{Average Wage for a junior geologist} = \$47/\textit{hour}\ [28]$$
$$S = \textit{Salary Savings Per Year} = W * (T_{Current} - T_{Saved}) = \$1243000$$

Moreover, assuming the utilization of open-source software during development, a gross margin of 100% is achievable which would then be invested back into Research and Development, helping improve the product and capture a larger market.

Due to the preceding analysis it is apparent that a successful solution must minimize manual intervention and reduce time spent on core logging. Therefore, this is the goal the following design process will try to achieve.

5.2 Legal Implications on Design

5.2.1 Avoiding Patent Infringement

Since the aim of this project is to deploy a fully functional system in industry, one must ensure no existing patents are infringed upon. During legal research, a 2006 patent was discovered that covered the photographing of a drill core from a fixed camera above, in order to extract alpha and beta angles [29]. While alarming at first, it was later realized that this patent focused on a specialized core box, and did not incorporate any computer vision. Further talks with the project's client company confirmed that the project was not at risk of infringing on the patent.

5.2.2 Following Applicable Standards

Again, since this project will be implemented in industry, it must conform to the applicable industry standards. A survey of standards indicated that the reference points of alpha and beta were crucial. For example, beta is always measured clockwise around the drill core [30]. Such standards were incorporated into the design of this system.

6 Design Process

6.1 Overview of Design Process

With the proper mathematical foundation now in hand, the design process can now be explained. For clarity, the design process in broken up into four distinct sections, illustrated in Figure 12 and described below.

[1] Obtained from testimony of a professional geologist Abdul Razique.

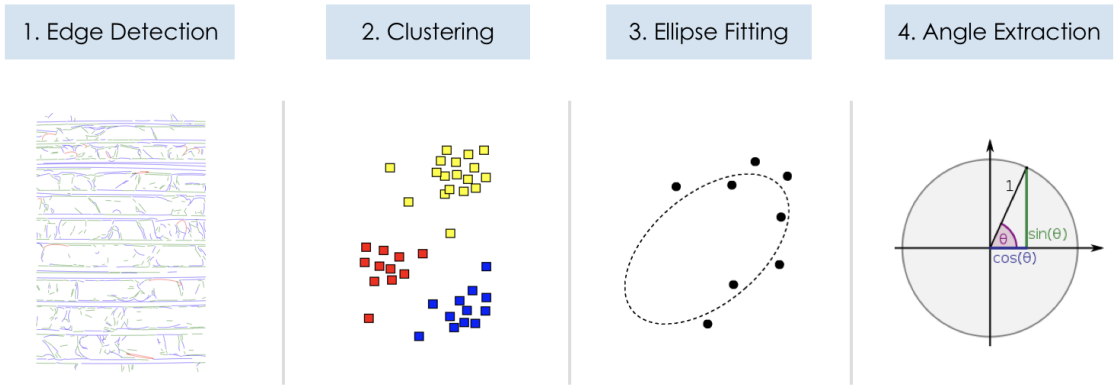


Figure 12: Major components of design process.

1. **Edge Detection**: Multiple edge detection algorithms will be explored in order to properly extract features from photos of drill core.
2. **Clustering**: To collect nearby features near a single fracture, various clustering algorithms will be explored.
3. **Ellipse Fitting**: As a precursor to alpha and beta angles, an ellipse must be fit to features clustered around each joint.
4. **Angle Extraction**: Using both trigonometry and knowledge of the drill core's orientation, alpha and beta will be calculated. Further, these angle pairs will be shown in a visualization tool.

6.2 Feature Detection

In order to accurately compute alpha and beta angles, all straight lines and elliptic arcs in a given image of drill cores need to be appropriately detected and extracted for analysis. In particular, joints, which are naturally occurring fractures in the drill core, need to be carefully selected. From above the core box these fractures take the shape of partial ellipses. To be concrete, the reader will note such features in Figure 13 below. The particular areas of interest are the regions in which the drill core samples break naturally along a joint.

Figure 13: Example drill core input with joints circled

Moreover, the drill core samples need to be clearly distinguishable from one another. Any automated solution should be able to identify the 11 different drill core samples in the given box of Figure 13, on top of identifying the number of fractures in each. In order to satisfy this, an automatic feature detection and extraction algorithm needs to be developed.

An emergent and promising method of feature detection that has been gaining traction in recent years are neural networks. In particular, convolutional neural networks have been shown to far outperform any classical means of feature detection [31]. However, due to the vast amounts of training data and parameter tuning needed for these algorithms, this method was deemed infeasible in context of this project. The remaining classical geometric feature detectors are generally classified into two categories: Hough-Based and edge chaining methods [2]. The first algorithm proposed in the latter category is the Canny edge detector developed by John F. Canny in 1986, due to its speed and popularity [32]. The algorithm can be summarized as follows:

1. Apply a Gaussian filter to smooth the image and remove noise
2. Find the intensity gradients of the image, apply non-maximum suppression to thin out the edges
3. Apply double thresholding to determine potential weak and strong edges
4. Suppress all other edges that are weak and not connected to strong edges [32]

This algorithm was implemented on a sample image of drill cores and the results can be observed in Figure 14.

Due to the poor performance of feature isolation of the Canny edge detector, alternate methods were pursued to address this problem. To be clear, the Canny edge detector is exceptional at finding edge pixels. However, this project is specifically interested in the geometrical representation of the

Figure 14: Before and after canny edge detection

edges in order to work with their slopes and intercepts. One possible approach is to loop through all of the pixels in the image and compute this information at the end via some predefined metric. However, this is clearly very inefficient and does not guarantee success in any general context. As a result, a mechanism that gives more weight to pixels that are already in a line (or any shape of interest) is desired. For this reason, the Hough Transform was considered next.

Hough-based algorithms implement variants of the Hough transform coupled with detection thresholds [33]. Although varying in implementation methods, the basic idea of the transform remains unchanged. The Hough Transform accumulates in an array the votes granted by the edge points to each candidate feature. The array points that exceed the threshold are then deemed detections. The parameters involved in this process are the detection thresholds as well as the quantization precision of the accumulator array [33] . These values are empirically tuned which means that they will directly impact the number of false detections in an image. Too much permissiveness will introduce too many false positives while over-restraining will cause false negatives. Nonetheless, due to its many promising benefits, the Hough Transform was applied to the example image as before and the results can be seen in Figure 15 . The Hough Transform relies on the Canny de-

Figure 15: Before and after Hough transform

tector to figure out the edge pixels and then uses voting techniques determined by the detection and quantization thresholds. The green lines then are the detected features. The reader will note that at least a third of the features fail to be detected by the algorithm. The reasons for this are due to the constraints mentioned previously. Namely, the transform requires empirical tuning of its parameters for different images. In general, since the drill core inputs will always be slightly different from each other, the algorithm would require tuning for every single input. Coupling

this fact with the time constraints of the project as well as the earlier economic considerations of automation; this method was deemed to be infeasible.

As a result, the natural direction of the algorithm selection process pointed in the direction of selecting a parameterless one. A few papers have attempted to address the parameter issue in the past decade. Specifically, the issue of detection thresholds. The Progressive Probabilistic Hough Transform stops the voting process for a certain feature when an excess is present in the accumulator array that reaches a certain level of statistical significance [34]. Whether something is an accident or not is determined by using a cut-off value on the detection threshold, on the probability that the observed excess of votes occurred randomly in a noisy image, thereby reducing false negatives. However, this approach is not scalable since the cut-off values are set for a predefined image size [34].

A promising parameterless approach known as the a contrario approach was developed by Desolneux in 2008 [4]. This technique automatically computes detection thresholds based on the Helmholtz perception principle, which informally states that there is no perception in white noise. This technique also uses the same principles in assessing accidents as the Progressive Probabilistic Hough Transform, however, the former statistical framework is able to control the overall number of false detections. The detection thresholds are accordingly self-tuned and as an added bonus in the idea of parameterless algorithms, the prior step of detecting edges (via Canny edge for example) becomes unnecessary with this approach. Using such a framework, Patraucean, Gurdjos, and Grompone developed the Ellipse and Line Segment Detector (ELSD), which eliminates the need for prior edge detection and parameter tuning [2]. An example result of the algorithm is seen below in Figure 16. In contrast to Figures 15 and 14 , Figure 16 successfully and clearly distinguishes individual features

Figure 16: Before and after ELSD

of interest. In particular, straight lines, and partial elliptic curves. For the mentioned reasons of parameterless inputs, successful geometric interpretation and convenience due to speed, this is the detector of choice going forward. Note that the sample size for the above quantitative analysis of missed features is limited largely to drill core samples. However, there have been numerous detailed quantitative assessments of such algorithms from which this information was extrapolated from. The reader is referred to the works of Gurdjos and Rafael for a lengthy and detailed quantitative analysis of the various performances of the Canny, Hough, and ELSD algorithms [4]. A weighted evaluation matrix can be seen in Table 1, where each section is scored out of 5, and the scoring is seen in Table 12 in Appendix A.

Table 1: Weighted Evaluation of Feature Detection

Algorithm	Edge Detection (Weight = 6)	Feature Detection (Weight = 10)	Implementation Difficulty (Weight = 3)	Parameter Tuning (Weight = 7)	Total
Canny	5	0	4	5	77
Hough	5	4	2	2	90
ELSD	2	5	3	4	99

6.3 Filtering of Features

The output from the Ellipse and Line Segment Detector can be seen in Figure 17

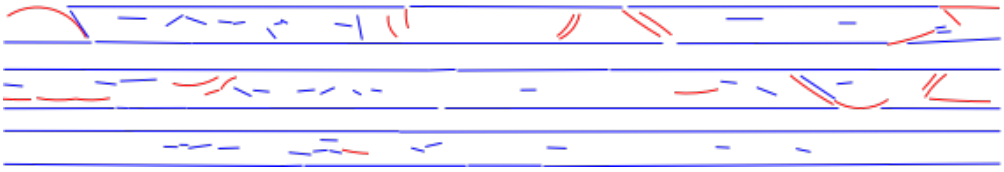


Figure 17: All detected features after ELSD

An image of drill core will typically contain more observable line segments than the ones of interest. To focus on the relevant segments, the algorithm filters them based on the following rules.

1. Any line that runs within 3°of parallel to the core box is deemed to be too steep, and not a joint fracture in the core itself.
   (a) Due to the dimensions of the core, any detected feature that is steeper than this threshold would vertically split the entire core, which would render all measurements erroneous.
2. Any feature less than or equal to 5 pixels in length are removed. This will remove any surface blemishes, or erroneous features.
   (a) If a feature is solely described by feature that is shorter than 5 pixels, the ellipse fitted to it would be susceptible to a high degree of error. During the user input stage, these joints should be manually selected and re-fitted.

The output of this filtering can be seen in Figure 18.

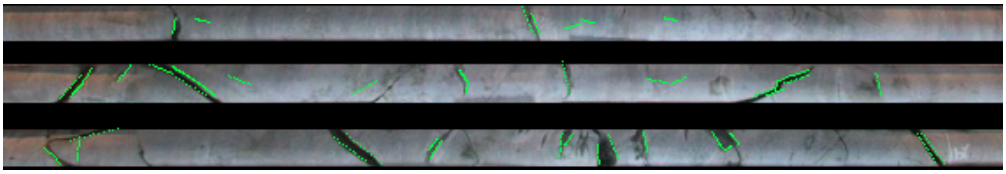


Figure 18: All detected features after ELSD + filtering

6.4 Clustering of Features

Once features within the image had been detected, ellipses needed to be fitted to all relevant features in order to produce measurements for alpha and beta. Joints would often be described by multiple detected features. In Figure 19, it is clear that the majority of the joints that needed to be measured were outlined by two or more individual features.

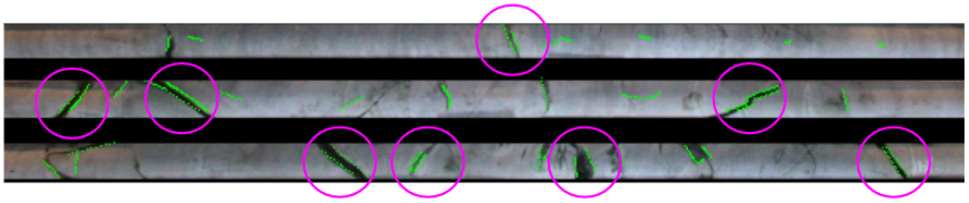


Figure 19: Joints described by multiple features.

This observation is what led to the search for a clustering algorithm. This would be a mapping from detected features to joints. This mapping would group together features that describe the same joint. Note that each individual green line represents a feature.

6.4.1 Literature Review for Clustering

In an article published in the International Journal of Computer Science and Information Technology, six different clustering algorithms were compared: K-Means, Farthest First, Expectation Maximization, Cobweb, Density Based, and Hierarchical [10]. The datasets across which these were compared are displayed in Table 13 in Appendix A.

The algorithms are compared according to the size of the dataset, the number of clusters, and the time taken to form those clusters. Since the number of clusters is a parameter that would eventually require optimization, this was not a relevant metric on which to compare results. Therefore, time was the primary metric that was considered. The timing results are shown in Table 2.

Table 2: Time Taken to Complete Clustering [10]

Algorithm	Dataset 1 (s)	Dataset 2 (s)	Dataset 3 (s)
K-Means	0.02	0.03	0.16
Expectation Maximization	1.98	124.95	966.41
Farthest First	0.01	0.03	0.09
Cobweb	0.06	0.72	1.17
Hierarchical	0.16	5.49	27.61
Density	0.02	0.11	0.17

Based on this metric, Farthest First performed the best, followed closely by K-Means. Since the user input of this solution can range anywhere from 5-20 seconds (Table 10), the 0.07s disparity between the two algorithms becomes nearly insignificant. K-Means also has the distinct advantage of being simple to implement in code (Section 11). Given these factors, K-Means clustering was the optimal choice for this technology.

6.4.2 Assessment of K-Means

The clustering results can be visualized in the sequence shown in Figure 20. Upon visual inspection, it can be deduced that the initial placement of centroids is not optimal. This is because there is a high density of centroids in one area, and a wide spread of data points that are set far from any centroid at all. As the iterations recalculate the partitions and centroid sets, an optimal partition is

being converged to. This convergence is guaranteed by the non-increasing function seen in Equation 4.2.3.

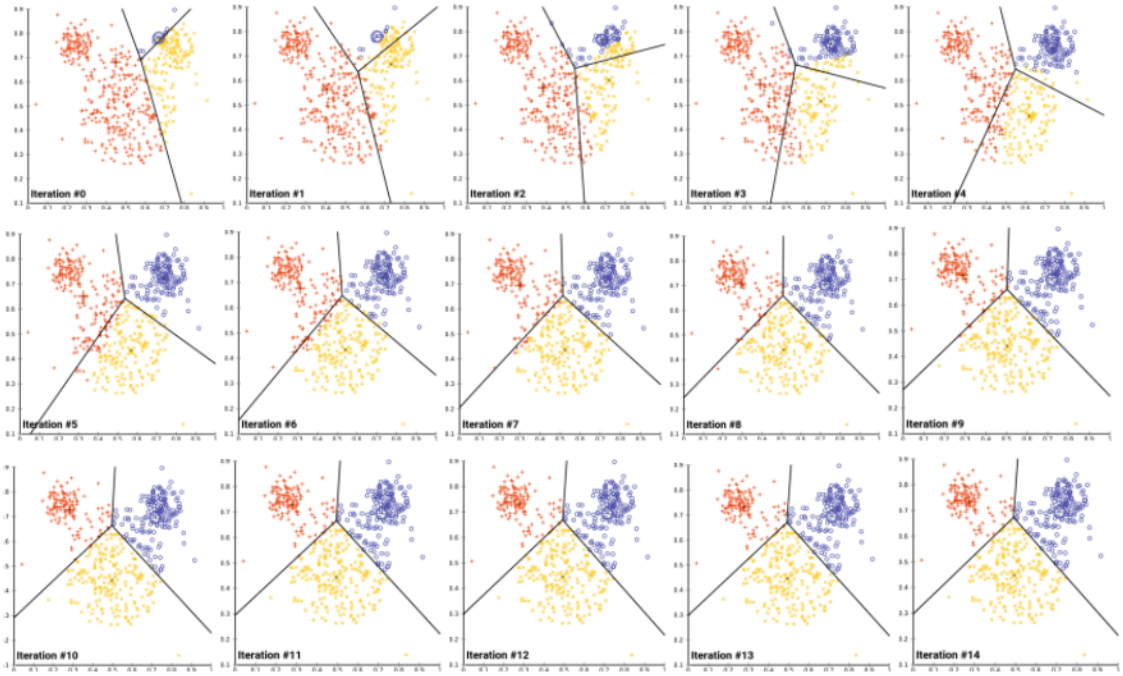


Figure 20: Demonstration of 2-Dimensional K-Means Clustering [6]

The benefits and drawbacks of K-Means clustering are listed below:

1. Advantages of K-Means
   (a) Low computational complexity of $O(t * K * N * d)$ [35]
      - $N = \#\ of\ data\ points$
      - $K = \#\ of\ clusters$
      - $t = \#\ of\ iterations$
      - $d = dimension\ of\ data$
   (b) Simple to visualize and understand
      - Open-source implementation available
   (c) Ability to choose number of clusters
      - Subject to optimization
   (d) Always converges to a minimum
2. Disadvantages of K-Means
   (a) Initialized with random centroids
      - Non-repeatable results
      - Susceptible to local minimum detection
   (b) Number of clusters is fixed
      - Inaccurate handling of outliers [7]

- Could perform poorly in noisy data

One aspect of the K-means clustering algorithm that is a weakness, is its inability to adjust the amount of clusters. As seen in Figure 21, when there are outlying points, adding another cluster would lead to a lower value of Equation 4.2.4. By requiring the outliers to associate with one of the existing clusters, the quality of clustering could be drastically reduced.

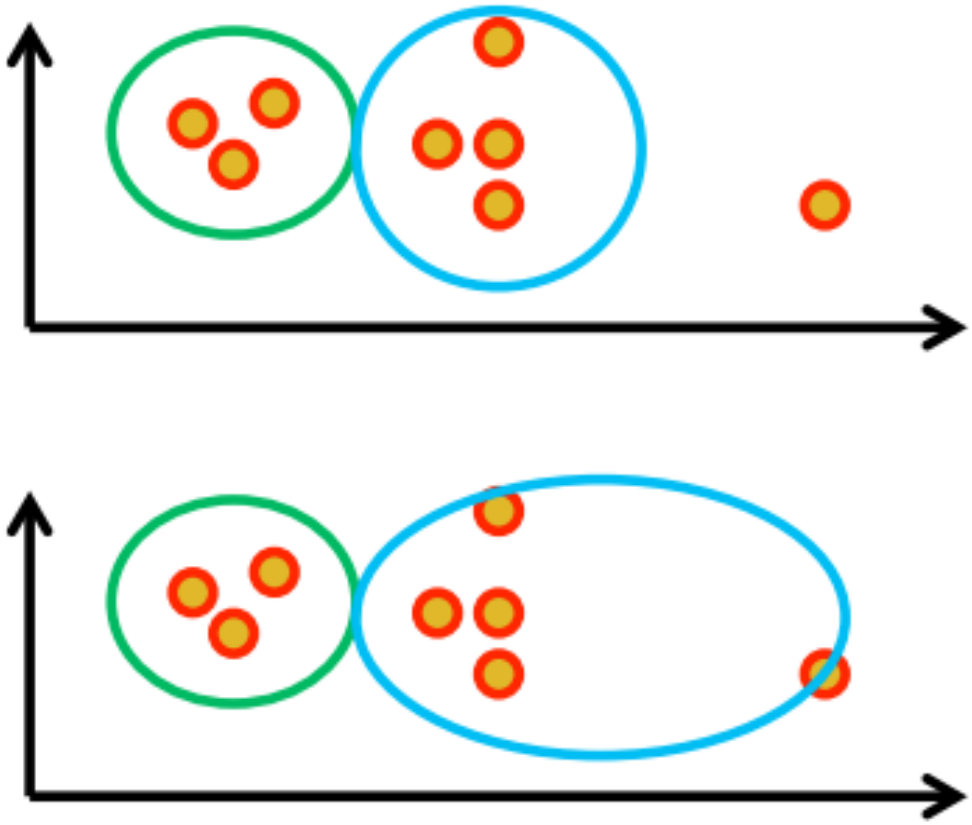


Figure 21: Outlier centroid association of K-Means Clustering [7]

Another weakness is the possibility of converging to a local minimum. While it is advantageous to know that the algorithm will converge as the iterations go to infinity, there is the possibility that this convergence is going to result in a local minimum. This behaviour is described in Figure 22. The seed pixels are randomly selected, and the resulting convergence produces a stable configuration, but visually it can be deduced that Equation 4.2.4 would be minimized further if each dense region of data had its own cluster associated to it.

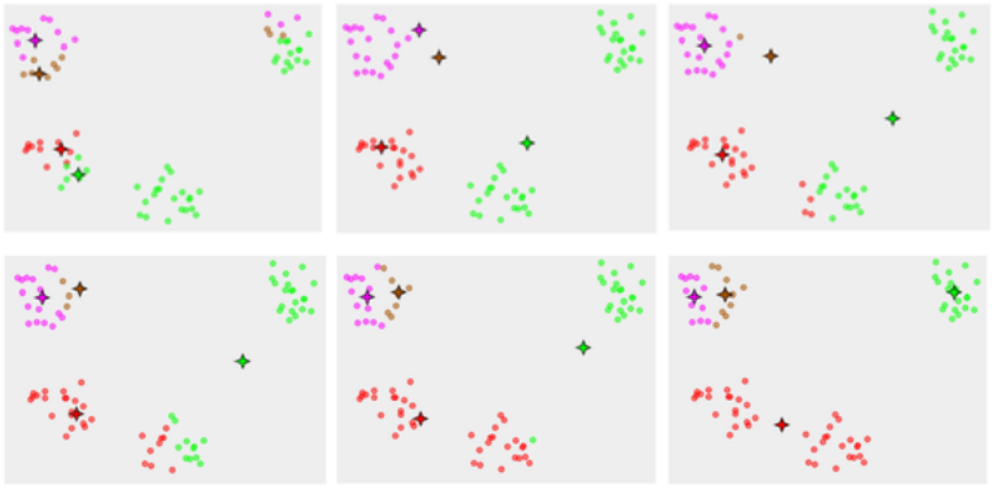


Figure 22: Example of K-Means settling on a local minimum. [6]

As described in Section 4.2, the K-means clustering accepts K as its only parameter. This provided the opportunity to optimize over this parameter. The following metric was used to assess quality.

$$K = \#\ of\ clusters\ per\ core$$
$$J = \#\ of\ clusters\ on\ a\ joint$$
$$C = 3 = \#\ of\ cores\ per\ box$$
$$Q = quality = \frac{J}{K * C} * 100$$

Values of K from 2 to 30 were tested and Figure 23 illustrates the associated quality rating. The optimal value was determined to be K = 18, which provided a 74% quality rating.

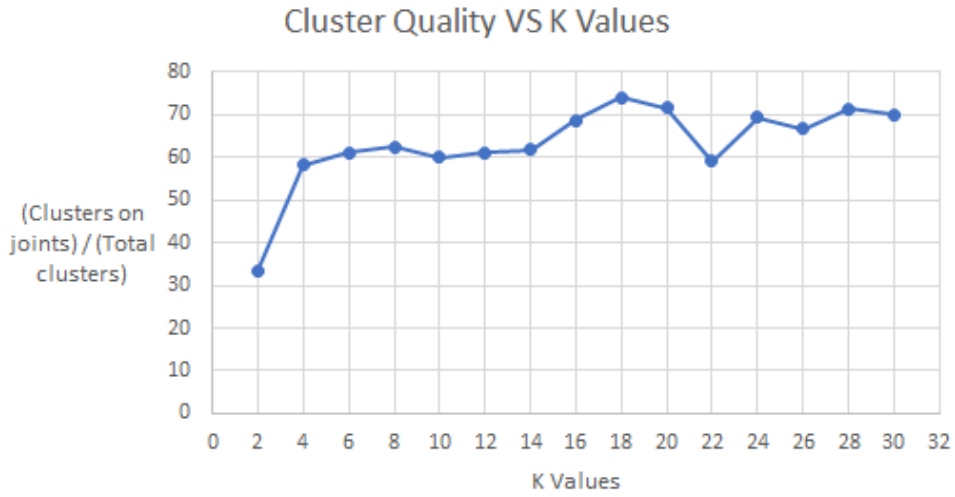


Figure 23: Cluster Quality VS K Value.

Since each core has fewer that 18 joints, there will often be multiple centroids located on one joint. This poses a problem since an ellipse should be fitted to each data cluster. This issue is illustrated in Figure 24.

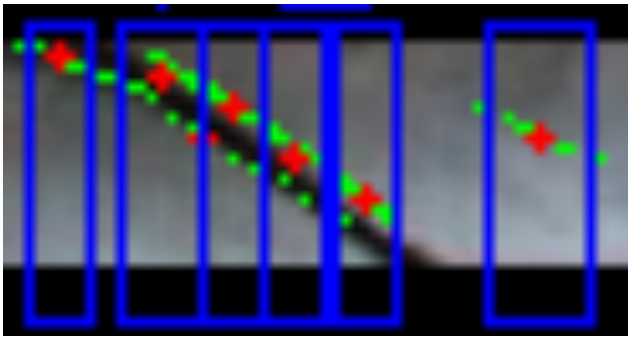


Figure 24: Multiple clusters will fall on a single joint.

In order to solve this lack of clarity, adjacent clusters are combined, and considered a valid dataset to be fitted with an ellipse. This is done by merging adjacent clusters. The results of this technique can be seen in Figure 25.

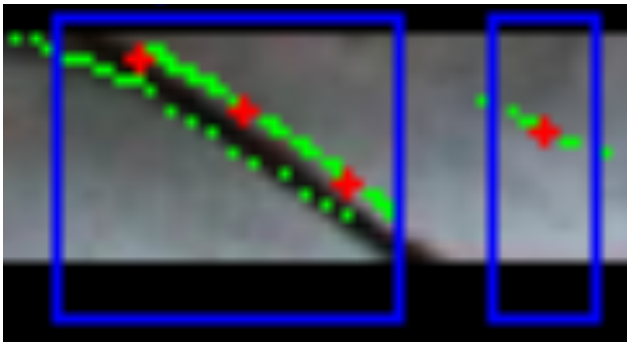


Figure 25: Adjacent clusters are merged.

The final result of the clustering algorithm can be seen in Figure 26. These combined clusters, outlined by a box, are now ready to have ellipses fitted to them.

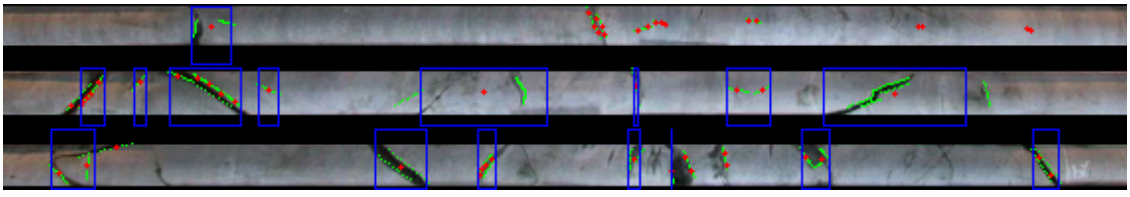


Figure 26: Results from feature clustering.

6.5 Ellipse Parameterization

Next, an ellipse will be fit to each cluster of features. This is because the ellipse is a precursor to finding alpha and beta. The drill core images have their pixels arranged in Cartesian coordinates

so this coordinate system was also used to parameterize ellipses. Here, an ellipse is described by a center point (x_0, y_0), a horizontal axis a, and a vertical axis, b, seen in Figure 27 and Equation 6.5.1.

$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} = 1 \tag{6.5.1}$$

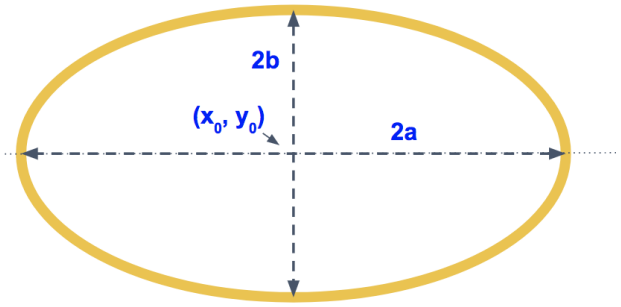


Figure 27: Diagram of Non-rotated Ellipse.

Furthermore, a rotated ellipse seen in Figure 28 and 6.5.1 in Cartesian coordinates is found by taking Equation 6.5.1, and rotating it by θ, where θ is angle of counter-clockwise rotation. Note that ellipse symmetry means θ falls within [0, 180]:

$$\frac{\Big((x - x_0)\cos(\theta) + (y - y_0)\sin(\theta)\Big)^2}{a^2} + \frac{\Big((x - x_0)\sin(\theta) - (y - y_0)\cos(\theta)\Big)^2}{b^2} = 1 \tag{6.5.2}$$

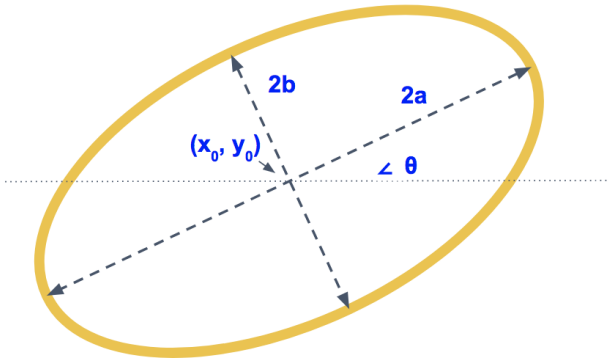


Figure 28: Diagram of Rotated Ellipse.

With a parameterization of rotated ellipses established in Cartesian coordinates, the fitting of an ellipse to a given dataset can now be discussed.

6.6 Ellipse Fitting with ODR

The ellipse is an undoubtedly familiar shape encountered in everyday life. Despite its simplicity, the particular technical intricacies of fitting the ellipse make it a problem that is not well suited

for traditional statistical techniques [36] . Two fundamental problems in ellipse fitting are (1) it falls into the class of errors-in-variables regression, and (2) the estimation problem is non-linear. One of the most common methods for fitting is the ordinary least squares (OLS) technique [37] and thus was the first technique considered. Through researching available information on the OLS, it became clear that it would not be sufficient to solve the ellipse fitting problem. This is mainly due to OLS's poor performance for errors-in-variables fitting, which leads to bias in the solution [38]. Further research revealed an algorithm called orthogonal distance regression (ODR) [38]. For ODR, the solution is found iteratively using a modified trust-region Levenberg-Marquardt method, which is an efficient and stable solution to the errors-in-variable fitting problem [38]. It is also specifically used to solve non-linear least squares problems so it addresses problem (2) of fitting ellipsis [39]. Additionally, the use of the trust-region algorithm allows ODR to perform much better than alternative procedures [17]. There are drawbacks to using ODR, however. Problems arise in the implementation given in 4.3.1 due to the approximations not ensuring that the constraint function $f(x_i;\boldsymbol{\beta})$ is exactly zero at the point where the orthogonal distance is implicitly computed [40]. It was decided that this paper was overly cautious because high accuracy was achieved by applying this algorithm to the constrained ellipse fitting problem to be defined in 6.7.4. In order to solve the ellipse fitting problem with ODR, the parameters derived in 4.3.1 are explicitly specified to coincide with the ellipse parameterization of 6.5.

$$[\beta_1, \beta_2, \beta_3, \beta_4, \beta_5] = [a, b, x_0, y_0, \theta]$$

$$f(x_i;\boldsymbol{\beta}) = \frac{\Big((h_i - \beta_3)\cos(\beta_5) + (v_i - \beta_4)\sin(\beta_5)\Big)^2}{{\beta_1}^2} + \frac{\Big((h_i - \beta_3)\sin(\beta_5) - (v_i - \beta_4)\cos(\beta_5)\Big)^2}{{\beta_2}^2} - 1$$

where $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)^T$ are the observed variables to be fit, and the variables $x_i = (h_i, v_i)$ are the horizontal and vertical distances from the origin, respectively. Also, $f(x_i;\boldsymbol{\beta})$ is a smooth function of its arguments. With $f(x_i;\boldsymbol{\beta})$ defined above, and using the L2-norm as in 4.3 in the derivation of the ODR algorithm, a linear rate of convergence can be achieved [41]. This convergence rate is achieved by choosing arbitrary initial conditions when initializing the algorithm, and it does not always result in convergence to the desired minimum. This will motivate further discussion on the choice of initial conditions.

6.7 Ellipse Fitting for Core Logging

6.7.1 Fitting Clustered Features

Next, an ellipse was fitted to each clustered set of detected edges. To do so, the edge detection data, as well as Equation 6.5.2, and initial conditions are passed onto Pythons SciPy ODR package, as outlined in Section 4.3.

The following vector is passed to the ODR as initial conditions:

$$[\beta_1, \beta_2, \beta_3, \beta_4, \beta_5] = [a, b, x_0, y_0, \theta] = [2 * CoreRadius, 5, x_{First}, y_{First}, 0]$$

Note that all variables passed to the ODR change during the search for an optimal ellipse, with the exception of x_0. This is because, by definition, the fitted ellipse must be aligned with the center of the core.

The above parameters are an approximation of the ellipse, based on what was known thus far. However, as one can see in Figure 29, instead of fitting an ellipse to an edge of the fault, the ODR package fits a single ellipse that incorrectly passes through both of the detected edges.

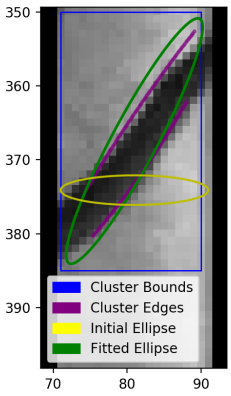


Figure 29: An incorrect ellipse fitting.

The resulting ellipse performance on the limited testing data is summarized in Table 3, and clearly indicates that the performance of this approach is lacking for two primary reasons. First, as seen in Figure 29, ellipses tend to erroneously fit both sides of a fault. Second, scratches and small irrelevant edges nearby a cleanly detected edge distort the proper shape of the ellipse and result in cases similar to that seen in Figure 30.

Table 3: Ellipse Fitting Performance on 13 Clusters

# of Proper Fits	# of Distorted Fits	# of Double Fits
1	9	3

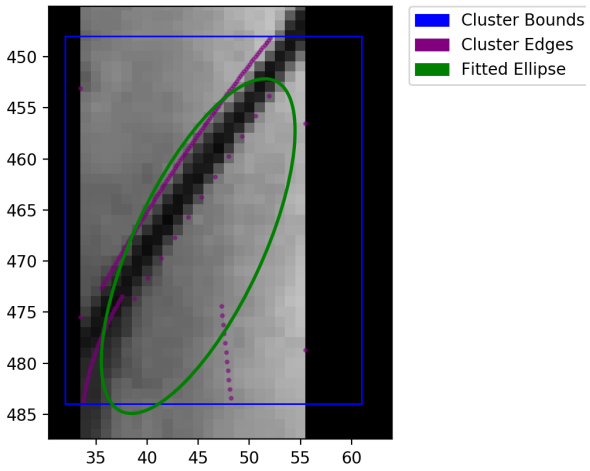


Figure 30: A distorted ellipse fitting.

The benefits of clustering features together were not immediately achieved. Instead, when tested, only 1 of the 13 clusters of features had a fitted ellipse that passed a visual quality inspection, seen in 3. While the sample size of 13 clusters within one image is not conclusive, it is indicative that a change in approach may be necessary. At this point. clustering features was put on hold.

6.7.2 Fitting Individual Features with Simple Initial Conditions

Instead of fitting an ellipse to a cluster of detected features, an ellipse will be fitted to each individual feature. Now, each feature will be fitted by an ellipse individually. Again, the following vector is passed to the ODR as initial conditions:

$$[a, b, x_0, y_0, \theta] = [2 * CoreRadius,\ 5,\ Core\ Center,\ y_{Average},\ 0]$$

When applying the ODR to individual features, instead of clusters, the ODR results are still insufficient. As seen in Figure 31, the ellipse fits the feature quite poorly. Upon further investigation, this type of fit was a systemic error. It is caused by when the ellipse parameters within the ODR converges to a local minimum instead of the global minimum.

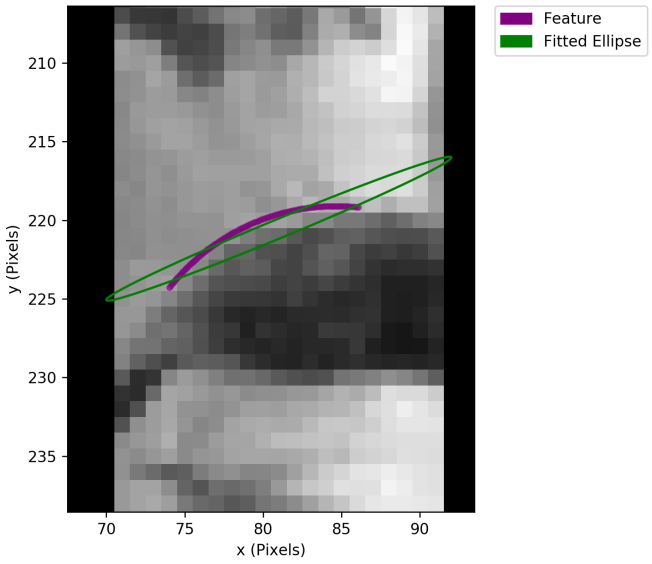


Figure 31: Local minimum reached.

Table 4 documents how many features were fitted correctly. Note that the vast majority of fitted ellipses are arriving at irrelevant local minimums instead of the desired global minimums.

Table 4: ODR Performance with Simple Initial Conditions

# of Proper Fits	# of Local Minimums
1	12

6.7.3 Analysis of Methods to Avoid Local Minimums

There are a few tools to explore in order to avoid having fitted ellipses arrive at local minimums. Preliminary research revealed strengths and weaknesses for each.

1. **Stochastic Gradient Descent**: The stochastic gradient (SG) algorithm behaves like a simulated annealing (SA) algorithm, in the sense that the learning rate of the SG is related to the temperature of SA [42]. The randomness or noise introduced by SG allows for a higher potential to escape from local minima to reach a better minimum, which in this case would be the global.

   (a) **Strength**: Faster convergence rates for larger data sets as opposed to regular gradient descent. Less likely to get stuck in local minima due to higher potential induced from randomness [43].

   (a) **Weakness**: Although the SG is less likely to get stuck in a local minimum than a conventional (batch) gradient descent, it cannot be guaranteed to always converge to a global minimum. Moreover, the SG is at its best when working with very large data sets by definition; since its main distinguishing feature is the fact that it updates only a

subset of training sample data from a training set as opposed to regular gradient descent which updates all data points for a given parameter [43]. In the context of this project, since the data sets will not be large enough to fully utilize this method, it will not be utilized.

2. **Simulated Annealing**: Implement simulated annealing, which finds an approximate global minimum instead of a precise local minimum
   (a) **Strength**: Incorporates random ascent to escape local minimums. [44].
   (b) **Weakness**: As a general meta-heuristic model, simulated annealing will often be outperformed by approaches that take advantage of any preexisting knowledge about the system. Further, for systems will few local minimums, such as the fitting of partial ellipses strictly along a drill core, simulated annealing is often unnecessarily computationally heavy [45].
3. **Multiple Initial Conditions**: Set multiple initial conditions for initial ellipse guesses and take whichever one converges with a minimum with the lowest error
   (a) **Strength**: Enforces systematic testing of multiple possible solutions, increasing the chances of finding the global minimum.
   (b) **Weakness**: While somewhat promising, this is a brute force and computationally heavy approach that is not conducive to the time saving constraint of the automated core logging problem.
4. **Smart Initial Conditions**: Set more intelligent initial conditions on the ellipse, making the initial ellipse more likely to converge to the global minimum.
   (a) **Strength**: Much is known about what values the final fitted ellipse should have. This makes smart initial conditions a natural next step.
   (b) **Weakness**: This approach assumes that starting near these initial conditions will help reach the global minimum.

In order to quantify the preceding analysis a weighted evaluation matrix was created, as seen in Table 5. Each category is out of 5, see Table 12 (Appendix A) for a further description of the scoring methodology.

Table 5: Weighted Evaluation of Ellipse Fitting Methods

Algorithm	Ease of Implementation (Weight = 6)	Likelihood to find Global Minimum (Weight = 10)	Computational Complexity (Weight = 8)	Total
Stochastic Gradient Descent	3	5	4	100
Simulated Annealing	3	5	2	84
Multiple Initial Conditions	4	5	2	90
Smart Initial Conditions	5	4	5	110

Given the immediate barriers associated with stochastic gradient descent and multiple initial conditions, and the drawbacks of simulated annealing, smart initial conditions is evidently the most suitable option. This is reflected in the weighted evaluation matrix.

6.7.4 Fitting Individual Features with Smart Initial Conditions

To implement smart conditions, one must take into account everything one knows about the detected feature and corresponding ellipse to be fitted:

1. Radius a: First off, are the two radii of the ellipse. Since the ellipse must cross from one side of the core to another, a way to approximate the first radius, a, would be to produce a line of best fit through the feature. Next, the line of best fit is extended across the core, from one edge to the other. The length of the resulting line is an approximation of radius a, as seen in Figure 32.

2. Radius b: Second, while one could use the curvature of the feature to estimate the second radius (or width) of the ellipse, b, training revealed that instead, a constant of b = 2 pixels performed adeptly.

3. x_0: Third, x_0 is left to be fixed at the center of the drill core.

4. y_0: Fourth, y_0 is set as the average y value of the feature to be fitted.

5. θ: Lastly, θ is approximated by using the slope of the line of best fit (LBF) that was used to approximate radius a, given by $\theta = \arctan(Slope\ of\ LBF)$

An example of a line of best fit through the feature, used to approximate a and θ can be seen in Figure 32.

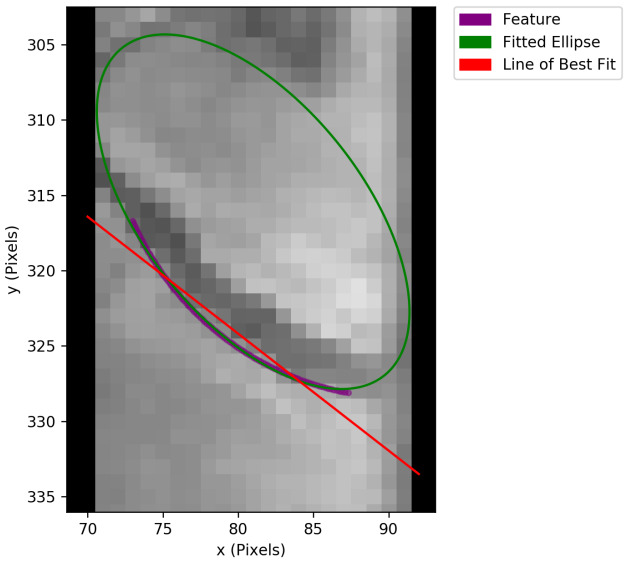


Figure 32: Line of best fit for radius a and θ approximation

With each parameter now intelligently approximated, the initial ellipse passed to the ODR will be much closer to the global minimum, and thus more likely to converge to it. Using the smart initial conditions and comparing them to performance of simple initial conditions, a sample feature typical of the performance difference can be seen in Figure 33.

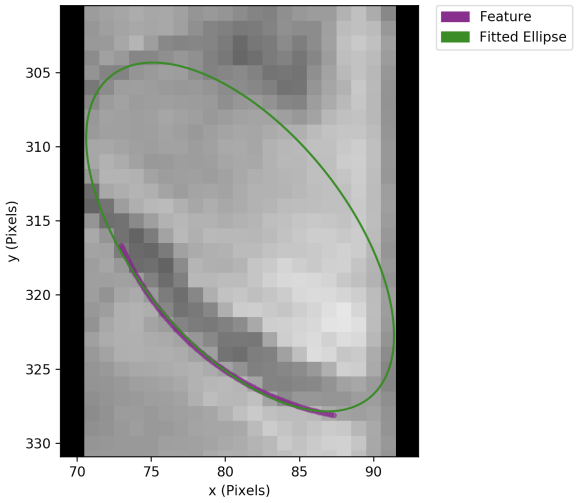


Figure 33: Properly fitted ellipse using smart initial conditions

In addition to fitting ellipses more accurately, the smart conditions naturally lead to less iterations of the ODR, decreasing its computation time by a factor of two. By running the ODR five times for both simple and smart initial conditions, a time comparison can be seen in Table 6.

Table 6: Effect of Smart Conditions on ODR Time

ODR Time for Simple Initial Conditions (ms)	ODR Time for Smart Initial Conditions (ms)
39.03	19.37

6.7.5 Biasing Alphas to be More Steep

Recall that alpha represents the slope of the fault lines. Imagine an engineer begins the excavation in a region believed to have gentle slopes, but in reality, the slope is much steeper. This means the stability and safety of the region are decreased.

To account for this real world implication, the initial angle of steepness given by the line of best fit will be biased to be more steep. In this way, the system will explore more steep ellipse parameters before gravitating towards more gently sloped ones. By biasing towards more steep alphas, the system will decrease the frequency that engineers and excavations are surprised by unstable landscapes.

6.7.6 Comparison of Multiple Ellipse Fitting Strategies

Table 7 summarizes the performance of each system at different stages in the design process.

Table 7: ODR Performance throughout Design Process

System	# of Correctly Fitted Features	# of Incorrectly Fitted Features
Clustered Features	1	12
Individual Features with Simple Initial Conditions	1	12
Individual Features with Smart Initial Conditions	10	3

6.8 Angle Extraction

With an ellipse approximately fitted to the joint face, alpha and beta can now be extracted.

6.8.1 Alpha Extraction

To extract alpha, *Height* is measured as the absolute value of the difference in y from the center of the ellipse (x_0, y_0) to the highest point on the ellipse, as seen in Figure 34. Note that *Height* is inversely proportional to α.

$$\alpha = \arctan\left(\frac{Core\ Radius}{Height}\right) \tag{6.8.1}$$

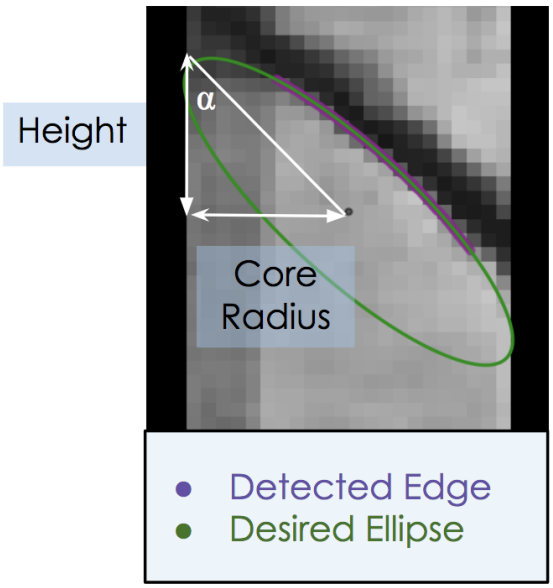


Figure 34: Dimensions required for alpha.

6.8.2 Beta Extraction

To calculate beta, one needs both the core radius, and the x offset. The x offset is the displacement in solely the x dimension of the lowest point on the ellipse. The first calculation is for beta*, which has not yet been properly measured against a reference frame, and only represents this internal angle between radius and x offset, given by Equation 6.8.2 and seen in Figure 35.

$$\alpha = \arccos\left(\frac{x\_Offset\ Radius}{Core\ Radius}\right) \tag{6.8.2}$$

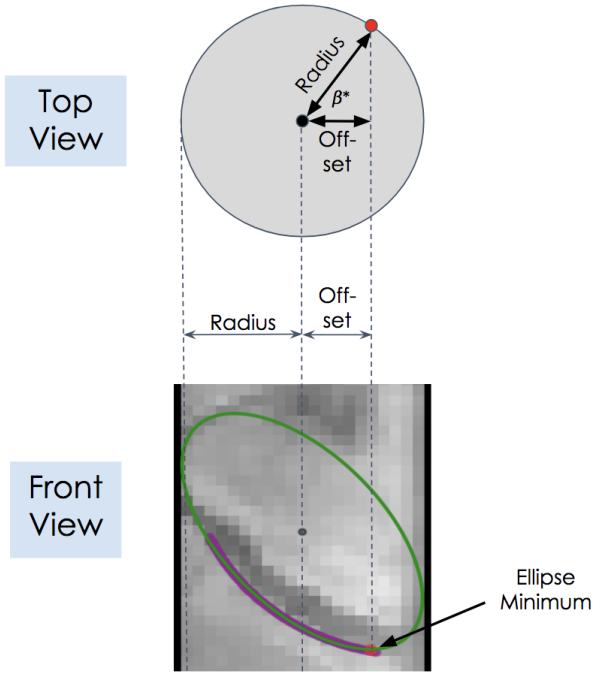


Figure 35: Dimensions required for beta.

After implementing this beta extraction method, it was noticed that all β^* values were in $[0°, 180°]$, when in practice, β falls within $[0°, 360°]$. The beta extraction process was revisited to detect the error and find a solution. After carefully outputting results alongside each image, the discrepancy was found. Seen in Figure 36, a single β^* value can have two potential true β angles. The industry standard of marking beta counterclockwise is followed, with 0 degrees facing the user [45].

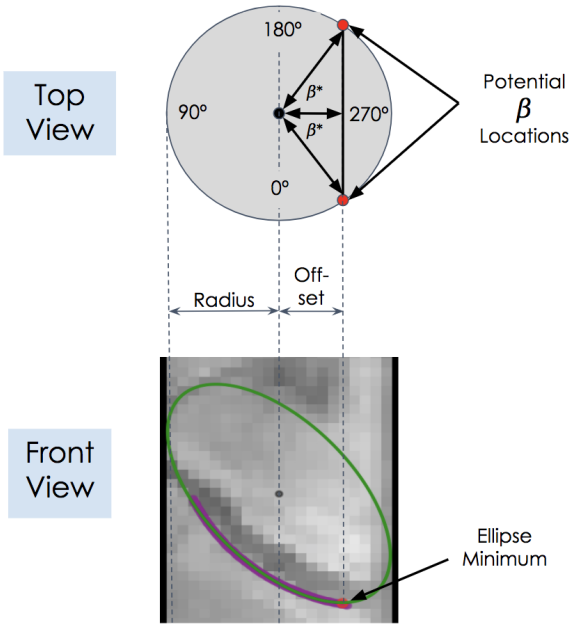


Figure 36: Ambiguous case for beta.

This now becomes a problem of deciphering whether β and the ellipse minimum lie at the front or the back of the core. By definition, and realized by observation, every time the data to be fitted (seen in purple) lies on the lower half of the fitted ellipse, as split along its longer axis, β will lie on the front of the core. Similarly, every time the data to be fitted (seen in purple) is on the upper half of the ellipse, beta will lie on the back of the core. This relationship is utilized by selecting the midpoint of the purple data and seeing whether it is above or below the longer axis of the ellipse. Using both the position of the purple data relative to the axis, and the β^* value, the true beta can be determined with the following relationships:

1. **Feature on Top of Ellipse**: $\beta = 270° - \beta^*$. Results in β in $[90°, 270°]$
2. **Feature on Bottom of Ellipse**: $\beta = (270° + \beta^*)\ mod\ 360$. Results in β in $[270°, 90°]$.

6.9 Visualization of Findings

For visualization purposes, the equations of the planes represented by alpha and beta will need to be calculated. With α, β, and the center point of an ellipse, two orthogonal vectors can be constructed. These two vectors are crossed to find a normal to the plane. Subsequently, an equation in 3 dimensions can be derived in the following form:

$$ax + by + cz = d \tag{6.9.1}$$

Using MatPlotLib in Python, 3D plots of cylinders mimicking the core, along with a plane representing each alpha and beta pair at the appropriate depth can be constructed, as seen in Figure 37. It is both this image, and a table of α, β, and depth, that is returned as the final output for the geologist.

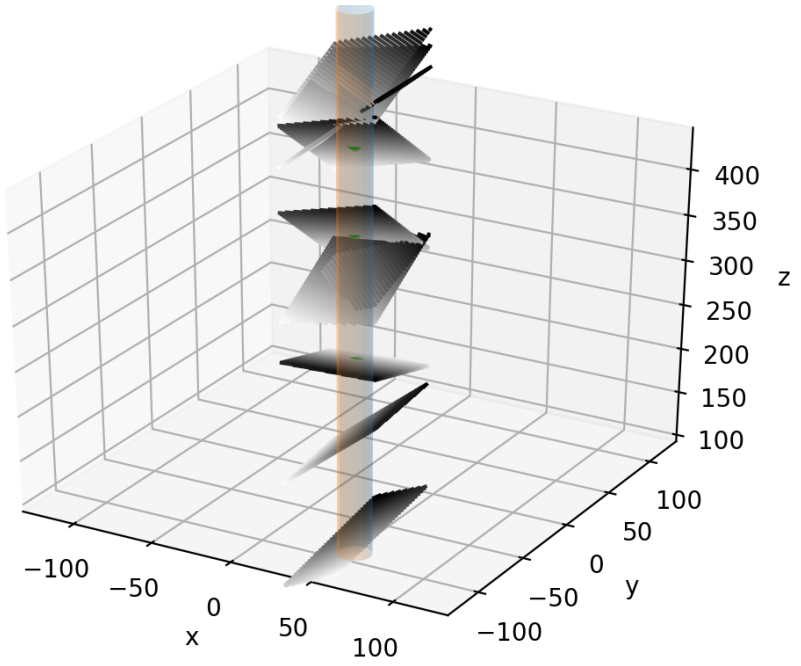


Figure 37: Visualization tool for geologists.

6.10 Summary of Design Process

This concludes the main portion of the design journey, which marks the automated extraction and visualization of α and β from an image of a core. A visual summary of the options considered, attempted, and selected, and how they all piece together, can be seen in Figure 38.

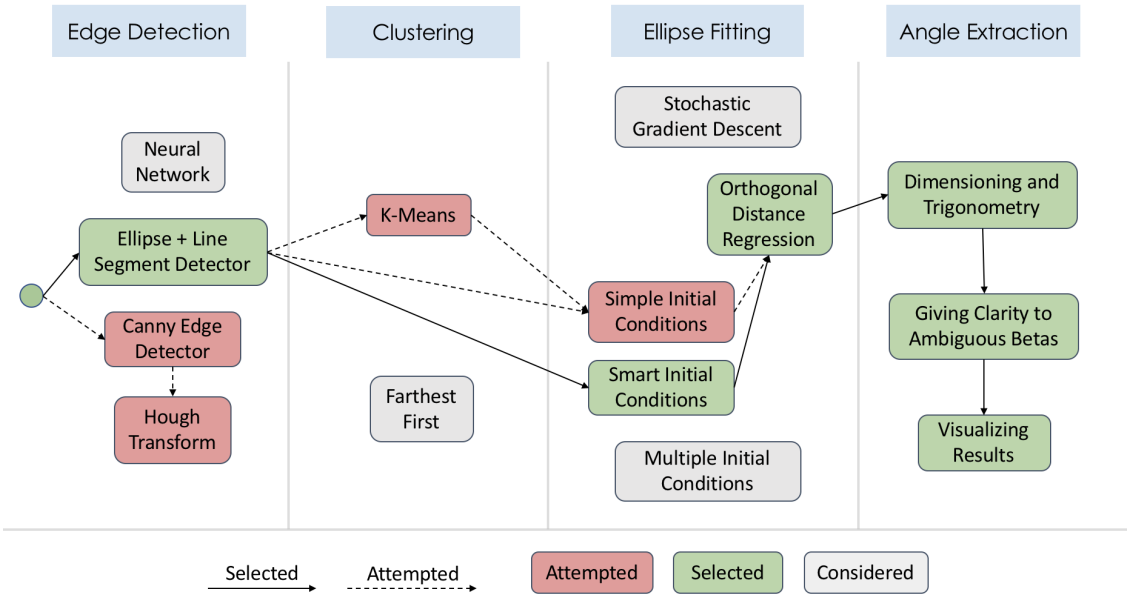


Figure 38: Visual of Design Process.

6.11 Collection of Testing Data

As mentioned, training and testing data was prohibitively expensive to incorporate intensively into the design process. However, several cores were logged manually in order to test the performance of the model. The following methodology was followed:

1. Visit Queen's University's Miller Hall collection of drill core samples
2. Measure alpha and beta angles with protractors and rulers
3. Document measured angles and record their general positions within the drill core
4. Photograph drill core from above, as squarely as possible.
5. Import photo and run final solution on it
6. Cross-reference solution angles with previously measured angles by matching positions within the photo

7 Results and Discussion

7.1 Results

Seen in Table 8 are both the hand-measured and system-measured α angles for each fault line that was hand-measured.

Table 8: Alpha Measurement Performance

Hand-Measured α	System Measured α	Difference
28.5°	29.6°	1.1°
79.3°	83.4°	4.0°
58.7°	58.0°	0.7°
53.0°	N/A	N/A
65.3°	62.9°	2.4°
31.1°	51.0°	19.9°

Seen in Table 9 are both the hand-measured and system-measured β angles for each fault line that was hand-measured.

Table 9: Beta Measurement Performance

Hand-Measured β	System Measured β	Difference
84.0°	105.5°	21.5°
345.6°	350.7°	5.08°
105.6°	99.4°	6.25°
31.2°	N/A	N/A
304.8°	314.0°	9.15°
12°	11.1°	0.9°

On these limited testing cores it is seen α and β are measured with average errors of 5.6° and 8.6°, respectively. Median errors for α and β were and impressive 2.4° and 6.25°, respectively. These averages fail to reflect the 6^{th} fault not detected by the system. Furthermore, of the 5 successfully detected fault lines, 4 had both the upper and lower images fitted properly, with measurements between the two being averaged for the final angles of that fault.

The time saved by this approach can be seen in Table 10.

Table 10: Time Comparison

Hand Measurements	System Measurements
35 minutes	User Selection: 10s Image Processing: 13s User Input: 5-20s **Total: 28-43s**

7.2 Discussion of Solution and Results

If more testing data was available, the α and β error would be characterized. These characterizations would be fed back into design decisions to both fine-tune and overhaul components of the solution.

7.2.1 Shortcomings

Several shortcomings and systemic errors arose from the system, seen in Figure 39.

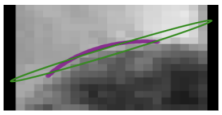
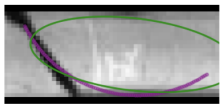
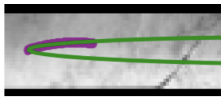
Flaw	Example	Impact	Potential Solution
Convergence on local minima		Incorrect α and β	Stochastic Gradient Descent
Missed or flawed edges		Distorted α and β	Image Prefiltering and Contrasting
Detection of mechanical breaks and scratches		Irrelevant α and β	User Input Override Option

Figure 39: Common shortcomings with examples.

Listed below are recommendations and possible solutions to the shortcomings outlined in Figure 39.

1. In order to avoid local minima, more high performance algorithms should be researched and implemented. This could include stochastic gradient descent, or simmulated annealing. Both of these are well-known optimization/minimization algorithms, but were not implemented in this project. Supplemental research in these areas could improve the robustness of this technology.

2. When joints are missed, α and β cannot be measured. A missed feature can be attributed to the ELSD algorithm not detecting or validating the particular region. Since the detection algorithm is initially searching for pixels with large gradient magnitudes, a missed fracture could be attributed to a lack of contrast in this region. Pre-filtering the image by increasing the contrast would improve the chances of a fracture being detected. The effects of this contrast increase can be seen in Figure 40. The algorithm would need to be run using this technique to determine if this is a beneficial feature to add.

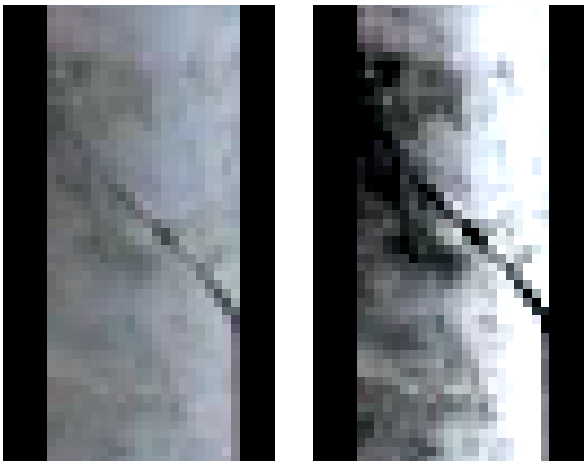


Figure 40: Initial image of a fracture (left) vs. the image with increased contrast (right)

3. When a fracture is wrongly identified, the resulting alpha and beta measurements can be wildly distorted. This mis-identification can be due to shadows on the core box, scratches, or mechanical breaks. These inconsistencies are very difficult for the algorithm to adjust for, so this is a beneficial area to ask for user input. Once the algorithm has identified regions that require ellipse fitting, the algorithm will prompt the user to confirm that all areas are correctly identified. Should a feature be distorted, the user simply needs to click the area, and then click to redraw the area of interest. This process is implemented currently, and is illustrated in Figure 41.

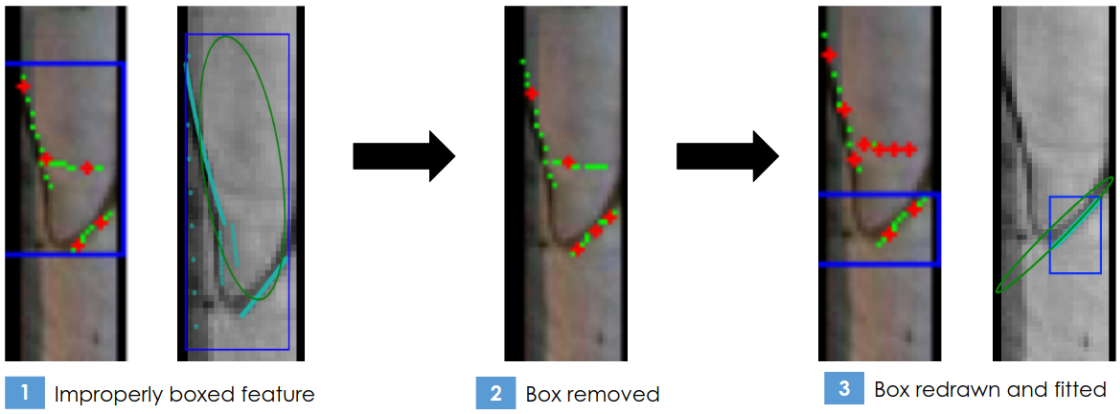


Figure 41: Demonstration of the user input stage for a wrongly identified fracture

The next steps will be to rerun the detection algorithm on the newly identified area. To avoid obtaining the same results, the newly identified area should also be subject to the contrast enhancement described above. This process is speculative as of now, and will require rigorous testing in order to validate the results, see Figure 42.

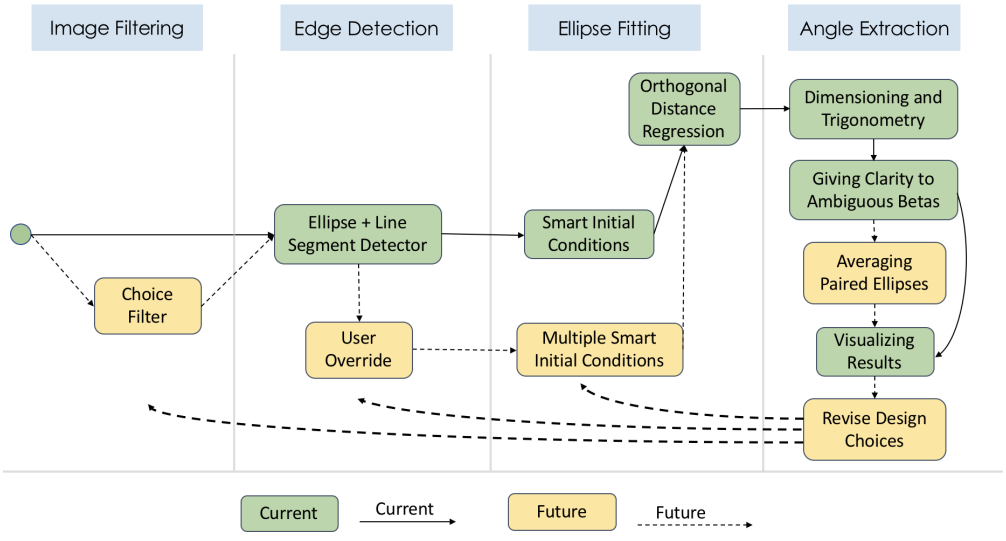


Figure 42: Visual of next steps.

7.2.2 Assumptions

In order to reduce the scope of the project, there were some simplifying assumptions made. These assumptions are listed in Table 11, along with recommendations on how to improve the functionality to accommodate them.

Table 11: Simplifying Assumptions and Next Steps Recommendations

Assumptions	Purpose	Next Steps
Camera directly above core box	This assumption allows precise location of the vertical edges of each segment of drill core.	These edges are characterized by relatively long, straight features, running essentially parallel to the edge of the image. To automate the detection of the edges of core segments, a process would need to be implemented to recognize these long features, and isolate the appropriate pixel values.
Fully trained operator	This was assumed throughout the development for simplicity, and to remove quality assurance measures from the scope.	A simple user interface, with helpful prompts should be implemented. This would include screening for erroneous inputs (eg. spaces in the output folder name), as well as an instruction manual. This manual should be easily available to any user in the form of a help option.
Orientation line is directly on top of the core	This allowed testing on any drill core that did not have a visible orientation line.	Similar to the core edges, resolving this assumption could involve the detection of a long vertical line, and using this pixel location as the basis for β values. Otherwise, this could also be an area for user input, where the user would be prompted to draw the orientation line.
Consistent lighting	This assumption was in place to assume that all intensity values could be expected to be similar on other core samples for future testing.	This could be resolved by increasing the brightness until the average pixel intensity is at the ideal level.

8 Triple Bottom Line

The core logging process is pivotal in ensuring the safety and prosperity of any mining operation. The following will outline the social, environmental, and economic implications of both successful

and unsuccessful core logging procedures.

8.1 Economic Considerations

To identify mineral deposits, mining companies need to invest a large amount of capital into exploration. The purpose of exploration is to discover if an area is worth mining and is suitable for building infrastructure around (e.g., mining shafts). An important piece in solving this puzzle is correctly logging core drilled from the ground. False alpha and beta measurements in this logging procedure can result in mischaracterizing the orientation and size of a mineral mineral deposit. As such, these angles are crucial in running a successful operation. With the average budget per mining company in 2016 totalling $4.4 million (aggregate across all totalling $6.9 billion), the cost of incorrectly surveying a site can result in a substantial loss [46]. In the 1970s, the mining company Sumitomo explored a site in Northwest British Columbia that should have been an exceptionally mineralised zone based on preliminary tests [47]. They used crude drilling techniques that resulted in them being unable to locate any significant deposits. Because of this, the company left the site thinking there was nothing to mine. After revisiting the site and improving upon previous mistakes, it is now considered one of the only Nickel-Copper-rich systems in the world, and likely to contribute to the already $7.3 billion dollar British Columbia mining industry [48].

Missing deposits are not the only economic risk that is faced by mining companies. In August 2014, the 40-metre-high tailings dam at the Mount Polley mine in British Columbia collapsed because of improper understanding of the alpha and beta angles of the ground underneath the dam [49]. This resulted in the drainage of 25-million cubic metres of contaminated sludge and mine waste into the surrounding lakes and rivers. This is the biggest mining spill in Canadian history, but Imperial Metals Corp, the parent company of Mount Polley, has not been fined, charged, or paid any significant amount to support the cleanup [50]. It was found that of the $67.4 million cost of cleanup, taxpayers were responsible for paying almost $40 million of it [51]. Economist Robyn Allan, who studied this case said, "B.C. taxpayers can expect to continue to pay for environmental harm because the provincial government refuses to take steps to implement meaningful policies that ensure that, when the polluter pollutes, it is the polluter and not B.C. taxpayers that pay" [51].

The economic ramifications of improperly exploring a mine can be severe, but there are also definitive economic benefits for implementing an improved core logging mechanism. For the IP owners (MTHE 493 group), they will see revenue by licensing the product to instrument and service providers (e.g., RockMass) and selling the product directly to mining companies. Instrument and service providers will see revenue by selling the product to mining companies, and mining companies will be able to recoup salary expenses. In Quebec alone, it is estimated that a total of $1.2 million could be saved annually, as calculated in Section 5 [25].

8.2 Societal Considerations

When the alpha and beta angles are used to plan excavation, the potential threat to public safety must be considered. For example, when road cuts are planned, the angles of the faults in that area will determine the risk of the wall collapsing, which puts traffic, pedestrians, and any roadside workers in jeopardy. As seen in Figure 43, approximately 12% of fatal mining accidents are a result of falls of ground, most of which were determined to be in zones of poor rock mass quality.
Negligence in understanding the geology of an area can be seen in many fatal accidents throughout history.

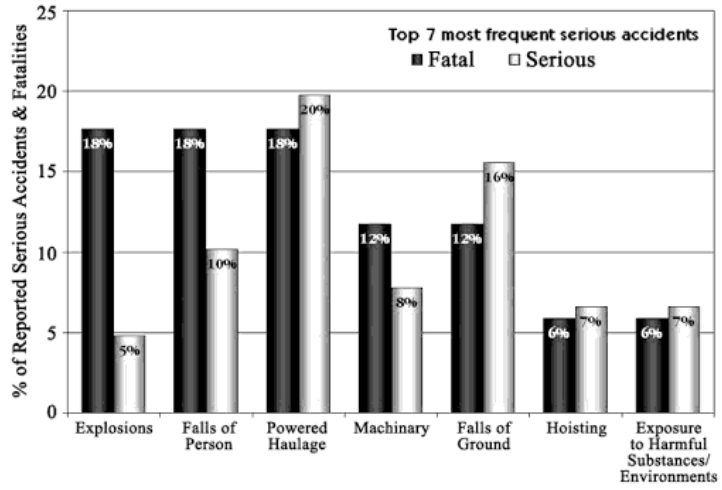


Figure 43: 12% of fatal mining accidents are a result of falls of ground [8].

In 2009, a metro line tunnel in Cologne, Germany collapsed due to faulty construction of one of the underground diaphragm walls [52]. The company responsible for the tunnels improperly factored for the irregular geological conditions in the surrounding area. This collapse lead to the death of two people as well as the loss of thousands of important historical documents, some being dated from the middle ages. Although this event was severe, the impacts of probing the Earth without correctly surveying the geology can have an even greater social impact.

In June of 2017 the collapse of a mine in Greece resulted in an 80 million cubic meter landslide [9]. The landslide was so large that it demolished a 200 person village. It's scale can be seen in Figure 44. Nobody was injured, but the farmers who lost their land permanently lost that source of income, and all the residents were forced to permanently evacuate the town. Had the mining company responsible for this more carefully considered the alpha and beta angles obtained during their survey of the land, they would have had the foresight to stop excavation before it lead to the acute geological conditions resulting in the landslide. The social impacts of precise structural assessment are a priority for anyone involved in geological projects, and must be thoroughly laid out before any work can be done.

Figure 44: 80 million cubic meter landslide in Greece swallows small town and farmland [9].

8.3 Environmental Considerations

When mining and geological projects are being planned, they often have serious environmental impacts due to their size. The information revealed by alpha and beta angles helps geologists understand where and how big a deposit is. Accurate measurements mean undesirable ground is more likely to be left alone and not mistakenly mined. This results in decreased disturbance and reduced emissions from using the land. The importance of this is amplified by considering the massive amounts of waste produced by mining operations.

When looking to quantify environmental impacts of mining, one must ensure their sources are trustworthy. Given the political nature of environmental considerations, many sources present propaganda-style arguments. For example, the One Green Planet organization publishes articles with title *"5 Ways Coal Mining is Destroying the Planet"* [53]. Such bias was avoided while sourcing objective information on the environmental impacts of mining.

Instead facts were drawn from WorldWatch, a fact-based organization that focuses transitioning to sustainability rather than criticizing current processes. According to WorldWatch, mining produces 2.7 billion tonnes of waste annually, most of which is toxic [54]. This is more than double what is produced worldwide by humans [55]. Additionally, more than 28 billion tonnes of material is stripped from the Earth annually. This is more than is moved by the natural erosion of all of Earths rivers [54]. Misinterpreting the size and location of a deposit is not the only concern, though.

Failure to understand the geometry of the Earth around the site can result in disasters like the one in Greece, mentioned above. Here 80 million cubic meters of land was turned to rubble, and the ground in the area remains unstable and at risk for more damage to occur. The mining disaster at Mount Pelley mine in B.C. had even more severe environmental repercussions, and these are still prevalent three years later. The tailings contained arsenic, copper, gold, manganese, nickel, lead, and vanadium, enabling many different toxic effects to cause harm to the rivers, lakes, and the people and animals that rely on these sources to survive [56]. Mining companies need to be conscious of the obligation to preserve the environment for posterity. The impact of mining disasters is undeniably evident and prevention of these events needs to become a majorpriority.

8.4 Ethical Considerations

The promise of automating the core logging process raises the important ethical question: what will happen to the geological engineers who previously had this job? Businesses across the globe are faced with this problem, and how they approach it can have impacts on individual working professionals and the world economy [57]. In some cases businesses do not need to replace workers. The workers can operate in a symbiotic relationship with the technology to further boost productivity. Although the problem appears mostly social, it can have a direct affect on the market value of companies using the technology. This is made clear in a recent study demonstrating that 83 percent of professional investors are more likely to buy stock in companies well known for social responsibility [58]. Furthermore, private investment brings the required capital, skills, and technology to enable profitability for mining companies [59]. Without these investments, desirable profit margins would be difficult to attain.
On the other hand, having automated core analysis software will make it possible for a given team of geologists to review more samples; leading to increased testing and faster throughput. In effect, this will support increased testing in situations where more detailed characterization is required

leading to more safety and environmental protection. Although the solution described in this project is completely automated, there are manual tasks surrounding the assembly and transfer of the technology between core boxes that can be done by a human. Additionally, in the next steps of this project, it is planned to include user input in order to inform the technology of any missed joints or mechanical breaks. When this is implemented it is certainly the case that geologists can operate alongside the technology to improve its performance. When considering the technology it is recommended that companies carefully consider the impact using this technology will have on their core values, workforce and investor relations.

9 Conclusions

This project attempts to remedy the inefficient process of manual geological core logging. Other attempts at a solution are generally too specialized and impractical, or require significant manual intervention. As a result, the work outlined in this paper attempts to distinguish itself by placing special emphasis on automation. In particular, a completely parameter-less feature detector in the ELSD is used to extract geometrical features in given core images with control over false positives. This approach is significantly faster than any competition as it eliminates the need for parameter tuning by the user. The resulting features are then pipelined into a sequence of clustering and fitting algorithms. The former ensures the elimination of faulty angles due to multiple similar but conflicting features; while the latter is a final precursor in describing the desired features within an elliptical framework. Upon the successful fitting of features, alpha and beta angles are computed by utilizing elementary trigonometry.

The industry average for the core logging process is approximately 30 minutes. This project effectively reduced this time to a range of 20-35 seconds while yielding an industrially accepted accuracy rating.

The entire project was limited due to the lack of sample training data, namely, drill core samples. This resulted in the elimination of commonly utilized methods such as neural networks. Moreover, time constraints led to the limited consideration of alternative design solutions, e.g., simulated annealing. In the future, more research would need to be conducted into a more robust solutions. This technology will then be able to help geological engineering companies make safer and smarter decisions.

10 Appendix A - Supporting Data

Table 12: Scoring metrics

Score	Metric
1	Not demonstrated
2	Below Expectation
3	Minimal Pass
4	Meets Expectation
5	Exceeds Expectation

Table 13: Clustering Datasets [10]

Dataset	# of Attributes	# of Instances
1	5	150
2	9	1253
3	9	2924

11 Appendix B - Code

11.1 Bash Wrapper Script - process.sh

```
#!/bin/bash

image=$(zenity --file-selection --title="
    Choose image")
outputDir=$(zenity --file-selection --
    directory --title="Choose output folder
    ")
defaultOutputFolder=$(date '+%d_%m_%Y_%H_%
    M_%S')
outputFolder=$(zenity --entry --title "
    Outlier Fraction" --text="Pick a name
    for the output folder." --entry-text=
    $defaultOutputFolder)
cd $outputDir
mkdir $outputFolder
cd $outputFolder
cp $image .
image_name=$(basename "$image")
separator="/"
new_image_path=$PWD$separator$image_name

python $HOME/mthe493/catkin_ws/src/
    processingDevice/src/process.py
    $new_image_path
```

11.2 Main Python Script - process.py

```
##--------------Libraries----------------

from PIL import Image
import numpy as np
from numpy import linalg as LA
import cv2
from matplotlib import pyplot as plt
from matplotlib import image as mpimg
from matplotlib.patches import Ellipse, Arc
    , Rectangle, Patch
from numpy.linalg import eig, inv
import subprocess as sub
import sys
import csv
import os
import math
import re
import shutil
from svg.path import Path, Line, Arc,
    CubicBezier, QuadraticBezier
import random
from scipy import odr
import scipy as sp
##--------------Subroutines----------------

def svgToCoords(svg_path):
  trunc = os.path.splitext(svg_path)[0]
  txt_path = trunc + ".txt"

  shutil.copyfile(svg_path, txt_path)

  in_path = txt_path
  out_path = trunc + "_coordinates.txt"
  core_1_path = trunc + "_core1.txt"
  core_2_path = trunc + "_core2.txt"
  core_3_path = trunc + "_core3.txt"

  in_f = open(in_path, 'r')
  out_f = open(out_path, 'w')
  core_1 = open(core_1_path, 'w')
  core_2 = open(core_2_path, 'w')
  core_3 = open(core_3_path, 'w')

  f = in_f.readlines()

  line_pattern = re.compile(r'<(?P<type>\w
      {4}) x1=\"(?P<x1>-?\d+\.?\d*)\" y1
      =\"(?P<y1>-?\d+\.?\d*)\" x2=\"(?P<x2
      >-?\d+\.\d*?)\" y2=\"(?P<y2>-?\d+\.?\d
      *)\".*')
  path_pattern = re.compile(r'<(?P<type>\w
      {4}) d=\"M (?P<x1>-?\d+\.?\d*),(?P<y1
      >-?\d+\.?\d*) A(?P<r1>\d+\.?\d*),(?P<
      r2>\d+\.?\d*) (?P<rotation>-?\d+\.?\d
      *) (?P<arc>\d{1}),(?P<sweep>\d{1}) (?P
      <x2>-?\d+\.?\d*),(?P<y2>-?\d+\.?\d*)
      \".*')

  for i, line in enumerate(f):

    if line.startswith("<line"):

      line_match = re.match(line_pattern,
          line)

      x1 = float(line_match.group('x1'))
      y1 = float(line_match.group('y1'))
      x2 = float(line_match.group('x2'))
      y2 = float(line_match.group('y2'))

      start = complex(x1,y1)
      middle = complex(int(x1+(x2-x1)/2),int
          (y1+(y2-y1)/2))
      end = complex(float(x2),float(y2))

      coords = getLineCoords(start, end)
      if coords != []:
        if middle.real in CORE1_XVALS:
          for i in coords: core_1.write(str(
              i)+",")
          core_1.write("\n")
        elif middle.real in CORE2_XVALS:
          for i in coords: core_2.write(str(
              i)+",")
          core_2.write("\n")
        elif middle.real in CORE3_XVALS:
          for i in coords: core_3.write(str(
              i)+",")
          core_3.write("\n")
        else:
          print "Error"
        out_f.write(str(coords))
        out_f.write("\n")

    elif line.startswith("<path"):

      path_match = re.match(path_pattern,
          line)
      x1 = float(path_match.group('x1'))
      y1 = float(path_match.group('y1'))
      r1 = path_match.group('r1')
      r2 = path_match.group('r2')
      rotation = float(path_match.group('
```

```
        rotation'))
      arc = int(path_match.group('arc'))
      sweep = int(path_match.group('sweep'))
      x2 = float(path_match.group('x2'))
      y2 = float(path_match.group('y2'))

      start = complex(x1,y1)
      radius = complex(float(r1),float(r2))
      middle = complex(int(x1+(x2-x1)/2),int
          (y1+(y2-y1)/2))
      end = complex(x2,y2)

      coords = getPathCoords(start, radius,
          rotation, arc, sweep, end)
      if coords != []:
        if middle.real in CORE1_XVALS:
          for i in coords: core_1.write(str(
              i)+",")
          core_1.write("\n")
        elif middle.real in CORE2_XVALS:
          for i in coords: core_2.write(str(
              i)+",")
          core_2.write("\n")
        elif middle.real in CORE3_XVALS:
          for i in coords: core_3.write(str(
              i)+",")
          core_3.write("\n")
        else:
          print "Error"
        out_f.write(str(coords))
        out_f.write("\n")

  in_f.close()
  core_1.close()
  core_2.close()
  core_3.close()
  out_f.close()

def getLineCoords(start, end):
  path = Line(start, end)
  coords = []

  if path.length() < MIN_LINE_LENGTH:
    return coords

  slope = getSlope(start, end)
  if abs(slope) > MAX_SLOPE:
    return coords

  for i in drange(0, 1, 0.1):
    coords.append(path.point(i))
  return coords

def getPathCoords(start, radius, rotation,
    arc, sweep, end):
  path = Arc(start, radius, rotation, arc,
      sweep, end)
  coords = []

  if path.length() < MIN_LINE_LENGTH:
    return coords

  slope = getSlope(start, end)
  if abs(slope) > MAX_SLOPE:
    return coords

  for i in drange(0, 1, 0.01):
    coords.append(path.point(i))

  return coords

def drange(start, stop, step):
    while start < stop:
    yield start
    start += step

def drawLine(coords, image, colour
    =[0,255,0]):
  for point in coords.split(", "):
    if point.startswith("["):
      point = point[2:-1]
    elif point.endswith("]"):
      point = point[1:-2]
    else:
      point = point[1:-1]

    point = complex(point)
    x = point.real
    y = point.imag
    try:
      image[y,x] = colour
    except(IndexError):
      continue
  return

def getSlope(start, end):
  rise = end.imag - start.imag
  run = end.real - start.real
  slope = rise/run
  return slope

# CLUSTER

def cluster_points(X, mu):
    clusters = {}
    for x in X:
    minDist = None
    bestmukey = None
```

```
    for i in enumerate(mu):
      dist = np.linalg.norm([x[0]-i[1][0],x
          [1]-i[1][1]])
      if minDist is None or minDist > dist:
        minDist = dist
        bestmukey = i[0]
      else:
        continue
    try:
      clusters[bestmukey].append(x)
    except KeyError:
      clusters[bestmukey] = [x]
    return clusters

def reevaluate_centers(mu, clusters):
    newmu = []
    keys = sorted(clusters.keys())
    for k in keys:
        newmu.append(np.mean(clusters[k],
            axis = 0))
    return newmu

def has_converged(mu, oldmu):
    return (set([tuple(a) for a in mu]) ==
        set([tuple(a) for a in oldmu]))

def find_centers(X, K):
    # Initialize to K random centers
    oldmu = random.sample(X, K)
    clusters = cluster_points(X, oldmu)
    mu = random.sample(X, K)
    while not has_converged(mu, oldmu):
        oldmu = mu
        # Assign all points in X to clusters
        clusters = cluster_points(X, mu)
        # Reevaluate centers
        mu = reevaluate_centers(oldmu,
            clusters)
    return(mu, clusters)

def getClusterLength(points):
  highest = None
  lowest = None
  for point in points:
    if highest is None:
      highest = point[1]
    elif point[1] > highest:
      highest = point[1]
    else:
      highest = highest

    if lowest is None:
      lowest = point[1]
    elif point[1] < lowest:
      lowest = point[1]
    else:
      lowest = lowest

  distance = highest - lowest
  return distance

def colourBoxes(boxes, image, xvals, core):

  for box in boxes:
    x1 = xvals[0]
    y1 = box[1]
    x2 = xvals[-1]
    y2 = box[0]
    for x in xvals:
      try:
        image[y1,x] = [255,0,0]
        image[y2,x] = [255,0,0]
      except IndexError:
        next

    y = min(y1,y2)
    while y < max(y1,y2):
      try:
        image[y,x1] = [255,0,0]
        image[y,x2] = [255,0,0]
        y += 1
      except IndexError:
        y += 1
        next

def colourClusters(clusters, image):
  for cluster in clusters:
    image[cluster[1],cluster[0]] = [0,0,255]
    image[cluster[1]+1,cluster[0]] =
        [0,0,255]
    image[cluster[1],cluster[0]+1] =
        [0,0,255]
    image[cluster[1],cluster[0]-1] =
        [0,0,255]
    image[cluster[1]-1,cluster[0]] =
        [0,0,255]

def saveBoxes(boxes, box_file):
  bf = open(box_file, 'w')
  for box in boxes:
    bf.write(str(box))
  bf.close()

def getMB(x1,y1,x2,y2):
  rise = y2 - y1
  run = x2 - x1
  slope = rise/run
  intercept = y1 - slope*x1
```

```
  return slope, intercept

def addBox2(event, x, y, flags, params):
  global added_boxes

  if event == cv2.EVENT_LBUTTONDOWN or
      event == cv2.EVENT_LBUTTONUP:
    if event == cv2.EVENT_LBUTTONDOWN:
      start = (int(x),int(y))
      print "START: " + str((x,y))
    if event == cv2.EVENT_LBUTTONUP:
      end = (int(x), int(y))
      print "END: " + str((x,y))
    added_boxes.append(start,end)

def addBox(event, x, y, flags, param):
  global added_box_points, adding

  if event == cv2.EVENT_LBUTTONDOWN:
    start = (x, y)
    added_box_points.append(start)
    adding = True

  elif event == cv2.EVENT_LBUTTONUP:
    end = (x,y)
    added_box_points.append(end)
    adding = False

def removeBox(event, x, y, flags, params):
  global removed_boxes

  if event == cv2.EVENT_LBUTTONUP:
    click = (int(x), int(y))
    print "CLICK: " + str((x,y))
    removed_boxes.append(click)



# OLD ELLIPSE FITTING

def fitEllipse(x,y):
    x = x[:,np.newaxis]
    y = y[:,np.newaxis]
    D =  np.hstack((x*x, x*y, y*y, x, y, np.
        ones_like(x)))
    S = np.dot(D.T,D)
    C = np.zeros([6,6])
    C[0,2] = C[2,0] = 2; C[1,1] = -1
    E, V = eig(np.dot(inv(S), C))
    n = np.argmax(np.abs(E))
    a = V[:,n]
    return a

def ellipse_center(a):
    b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a
        [4]/2, a[5], a[0]
    num = b*b-a*c
    x0=(c*d-b*f)/num
    y0=(a*f-b*d)/num
    return np.array([x0,y0])


def ellipse_angle_of_rotation( a ):
    b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a
        [4]/2, a[5], a[0]
    return 0.5*np.arctan(2*b/(a-c))


def ellipse_axis_length( a ):
    b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a
        [4]/2, a[5], a[0]
    up = 2*(a*f*f+c*d*d+g*b*b-2*b*d*f-a*c*g)
    down1=(b*b-a*c)*( (c-a)*np.sqrt(1+4*b*b
        /((a-c)*(a-c)))-(c+a))
    down2=(b*b-a*c)*( (a-c)*np.sqrt(1+4*b*b
        /((a-c)*(a-c)))-(c+a))
    res1=np.sqrt(up/down1)
    res2=np.sqrt(up/down2)
    return np.array([res1, res2])

def ellipse_angle_of_rotation2( a ):
    b,c,d,f,g,a = a[1]/2, a[2], a[3]/2, a
        [4]/2, a[5], a[0]
    if b == 0:
        if a > c:
            return 0
        else:
            return np.pi/2
    else:
        if a > c:
            return np.arctan(2*b/(a-c))/2
        else:
            return np.pi/2 + np.arctan(2*b/(
                a-c))/2

def fitToBox(boxes, image_path, f, xvals):

  img=mpimg.imread(image_path)
  imgplot = plt.imshow(img, cmap='gray')

  fig = plt.gcf()
  ax = fig.gca()

  for box in boxes:
    top = int(box[1])
    bot = int(box[0])
    left = xvals[0]
    right = xvals[-1]
```

```
rec = Rectangle(
    (left, top),  # (x,y)
    right-left,        # width
    bot-top,        # height
    fill = False,
    edgecolor="red"
  )
ax.add_patch(rec)

x = []
y = []
xt = []
yt = []

f.seek(0)
for line in f.readlines():
  coords = []
  line.rstrip(',\n')
  for point in line.split(","):
    if point != '\n':
      coords.append(complex(point))
  for point in coords:
    x.append(point.real)
    y.append(point.imag)
    if point.real > left and point.real
        < right:
      if point.imag > bot and point.imag
          < top:
        xt.append(point.real)
        yt.append(point.imag)
if xt == [] or yt == []: continue
plt.scatter(x, y, s=1, c="green", alpha
    =0.5)
plt.scatter(xt, yt, s=1, c="blue", alpha
    =0.5)
xt = np.asarray(xt)
yt = np.asarray(yt)

el = fitEllipse(xt,yt)
center = ellipse_center(el)
phi = ellipse_angle_of_rotation(el)
axes = ellipse_axis_length(el)

elFit = Ellipse(xy=(center[0], center
    [1]), width=2*axes[1], height=2*axes
    [0], angle=phi*180/math.pi,
    edgecolor='y', fc='None', lw=1)

ax.add_patch(elFit)


plt.show()
```

```
# NEW ELLIPSE FITTING

def getCoords(coordsFile):
    with open(coordsFile, 'r') as f:
        reader = csv.reader(f)
        coords = list(reader)
    return processCoords(coords)

def processCoords(coords):
    procX = []
    procY = []
    for feat in coords:
        featX = []
        featY = []
        for point in feat:
            if point.startswith("["):
                point = point[2:-1]
            elif point.endswith("]"):
                point = point[2:-2]
            else:
                point = point[2:-1]
            point = complex(point)
            featX.append(point.real)
            featY.append(point.imag)
        featX = np.asarray(featX)
        featY = np.asarray(featY)
        procX.append(featX)
        procY.append(featY)
    return [procX, procY]

def getImage(imageFile):
    plt.figure(num=None, figsize=(8, 6), dpi
        =120, facecolor='w', edgecolor='k')
    img=mpimg.imread(imageFile)
    imgplot = plt.imshow(img, cmap='gray')
    fig = plt.gcf()
    ax = fig.gca()
    return img

def plotVlines(vLines):
    for v in vLines:
        plt.vlines(x=v, ymin=0, ymax = 569,
            color = 'xkcd:sky blue')
        return

def fitAll(xCoords, yCoords, vLines, target
    , disp):
    totalErr = 0
    descTable = []
    descTable.append(['xMid', 'yMid', 'alpha
        ', 'Nbeta', 'feature No'])
    for i in range(0,len(xCoords)):
        xFeat = xCoords[i]
        yFeat = yCoords[i]
```

```
        if i in target:
            print('Feature: ' + str(i))
            description = fitOne(xFeat,yFeat
                ,vLines, disp, i)
            description.append(i)
            descTable.append(description)
    return descTable

def fitOne(xFeat, yFeat, vLines, disp,
    featNo):
    #plt.figure(num=None, figsize=(8, 6),
        dpi=120, facecolor='w', edgecolor='k
        ')
    imgplot = plt.imshow(img, cmap='gray')
    plt.scatter(xFeat, yFeat, s=12, c="
        purple", alpha=0.5) # C
    plt.title('Feature ' + str(featNo))
    bounds = whichCore(xFeat[0], vLines)

    #plt.scatter(xFeat[0], yFeat[0], s=10, c
        ="pink", alpha=0.5)
    lbf = np.polyfit(xFeat, yFeat, 1)
    radAppx = getRadAppx(lbf,bounds[0],
        bounds[1])
    [chi, err] = ODRregress(xFeat, yFeat,
        radAppx, bounds, lbf[0])
    plt.scatter(chi[2], chi[3], s=8, c="k",
        alpha=0.5) # C
    peaks = np.asarray(getPeaks(chi))
    depth = getDepth(bounds,peaks[0])
    trueRad = getTrueRad(peaks, depth)
    growth = trueRad/max(chi[0:2])*100
    alpha = getAlpha(trueRad,bounds)
    beta = getBeta(peaks[0],bounds)
    [newBeta, quadrant] = correctBeta(xFeat,
        yFeat,peaks[0:2],peaks[2:4],beta)

    xMid = round(np.mean([peaks[0],peaks
        [2]]),2)
    yMid = round(np.mean([peaks[1],peaks
        [3]]),2)
    return [xMid, yMid, round(alpha,2),
        round(newBeta,2)]

def correctBeta(xFeat,yFeat, bottom, top,
    beta):
    point = [np.mean(xFeat),np.mean(yFeat)]
    #plt.scatter(point[0], point[1], s=10, c
        ="pink", alpha=0.5) # C
    x = bottom[0]
    xMid = np.mean([x,top[0]])
    if beta == -1:
        return [-1, 'fail']
    elif lineSolve(point, bottom, top): #
        Back half.
        return [270 - beta, 'back']
    elif x < xMid: # Front left.
        return [(beta - 90) % 360, 'front
            left']
    else: # Front right.
        return [270 + beta % 360, 'front
            right']

def lineSolve(point, bottom, top):
    a = (top[1] - bottom[1])/(top[0] -
        bottom[0])
    b = bottom[1]- a*bottom[0]
    if point[1] > lineF(a,b,point[0]):
        return False
    else:
        return True

def lineF(a,b,x):
    return a*x + b

def plotEll(beta, color):
    # beta = [a, b, xc, yc, alpha] FYI:
        contents of beta.
    c = max(beta[0],beta[1])
    xlist = np.linspace(beta[2]-2*c, beta
        [2]+2*c, 1000)
    ylist = np.linspace(beta[3]-2*c, beta
        [3]+2*c, 1000)
    X,Y = np.meshgrid(xlist, ylist)
    F = (( (X-beta[2])*math.cos(beta[4])+(Y-
        beta[3])*math.sin(beta[4]) )**2)/(
        beta[0]**2) \
        + (( (X-beta[2])*math.sin(beta[4]) -
            (Y-beta[3])*math.cos(beta[4]) )
           **2)/(beta[1]**2) - 1
    contourObj = plt.contour(X, Y, F, [0],
        colors = color, linestyles = 'solid
        ')
    contourObj = contourObj.collections[0].
        get_paths()[0]
    return contourObj.vertices

def plotLine(a,b, x_range):
    x = np.array(x_range)
    formula = str(a) + '*x + ' + str(b)
    y = eval(formula)
    plt.plot(x, y,'r')

def getRadAppx(lbf,x1,x2):
    a = lbf[0]
    b = lbf[1]
    x1 = 33
    x2 = 55
```

```
    y1 = a*x1 + b
    y2 = a*x2 + b
    p1 = np.array([x1,y1])
    p2 = np.array([x2,y2])
    radAppx = np.linalg.norm(p1-p2)
    return(radAppx/2)

def whichCore(x,vLines):
    mids = [ np.mean(vLines[0:2]), np.mean(
        vLines[2:4]), np.mean(vLines[4:])]
    one = abs(x - mids[0])
    two = abs(x - mids[1])
    three = abs(x - mids[2])
    closest = min(one, two, three)
    if one == closest:
        return vLines[0:2]
    elif two == closest:
        return vLines[2:4]
    else: # 3 is closest
        return vLines[4:]

def ODRregress(xFeat, yFeat, radAppx,
    bounds, slope):
    xc = np.mean(bounds)
    yc = np.mean(yFeat)
    xx = np.array([xFeat, yFeat])
    mdr = odr.Model(f, implicit=True)
    mydata = odr.Data(xx, y=1)
    betaGuess = [radAppx, 2, xc, yc, math.
        atan(slope)]
    #plotEll(betaGuess, 'y') # Initial guess
        . Yellow
    fixed =       [  0  , 1 , 0 , 1 , 1 ] #
        Holds some variables constant.
    myodr = odr.ODR(mydata, mdr, beta0 =
        betaGuess, ifixb = fixed)
    myoutput = myodr.run()
    return [myoutput.beta, myoutput.
        sum_square]

def f(B, x):
    return (( (x[0]-B[2])*math.cos(B[4])+(x
        [1]-B[3])*math.sin(B[4]) )**2)/(B
        [0]**2) + (( (x[0]-B[2])*math.sin(B
        [4]) - (x[1]-B[3])*math.cos(B[4]) )
        **2)/(B[1]**2)-1.

def getPeaks(chi):
    verts = plotEll(chi,'g') # Fitted.
    peaks = getPeaksSub(verts)
    return peaks

def getPeaksSub(verts):
    xs = []
    ys = []
    xLow = 0
    yLow = 0
    xHigh = 700
    yHigh= 700
    for point in verts:
        xs.append(point[0])
        ys.append(point[1])
        if point[1] > yLow:
            xLow = point[0]
            yLow = point[1]
        if point[1] < yHigh:
            xHigh = point[0]
            yHigh = point[1]

    #plt.scatter(xLow, yLow, s=20, c="red",
        alpha=0.5)
    #plt.scatter(xHigh, yHigh, s=30, c="red
        ", alpha=0.5)
    return [xLow, yLow, xHigh, yHigh]

def getDepth(bounds, xCoord):
    center = np.mean(bounds)
    radius = (bounds[1]-bounds[0])/2
    depth = (radius**2 - (xCoord-center)**2)
        **(1/2)
    return(depth)

def getTrueRad(peaks,depth):
    p1 = np.asarray([peaks[0], peaks[1],
        depth]) # [x, y, z]
    p2 = np.asarray([peaks[2], peaks[3], -
        depth])
    return np.linalg.norm(p1-p2)/2

def getAlpha(trueRad, bounds):
    coreDiam = 1.0*bounds[1]-bounds[0]
    faceDiam = 2*trueRad
    try:
        alpha = math.asin(coreDiam/faceDiam)
            /math.pi*180
    except ValueError:
        alpha =-1
    return alpha

def getBeta(botX, bounds):
    coreRad = (1.0*bounds[1]-bounds[0])/2
    print(coreRad)
    xDisplace = botX - np.mean(bounds)
    try:
        beta = math.acos(xDisplace/coreRad)/
            math.pi*180
    except ValueError:
        beta = -1
```

```
    return beta


def mother(coordsFile, imageFile, vLines,
    target, display):
    [xCoords, yCoords] = getCoords(
        coordsFile)
    #ax = getImage(imageFile)
    #plotVlines(vLines)
    descTable = fitAll(xCoords, yCoords,
        vLines, target, display)
    edges = Patch(color='purple', label='
        Cluster Edges')
    guessed = Patch(color='yellow', label='
        Initial Ellipse')
    fitted = Patch(color='green', label='
        Fitted Ellipse')
    bounds = Patch(color='blue', label='
        Cluster Bounds')
    plt.legend(handles=[edges, fitted],
        bbox_to_anchor=(1.05, 1), loc=2,
        borderaxespad=0.)
    plt.xlabel('x (Pixels)')
    plt.ylabel('y (Pixels)')
    return descTable



def mergeBoxes(boxes):
  changes = False
  new_boxes = []
  to_remove = []
  for i,ref_box in enumerate(boxes):
    if i in to_remove: next
    for j,comp_box in enumerate(boxes):
      if i == j: next
      if ref_box[0] < comp_box[1]+SNAP_DIST
          and ref_box[0] > comp_box[0]:
        changes = True
        ref_box[0] = comp_box[0]
        to_remove.append(j)
      elif ref_box[1] > comp_box[0]-
          SNAP_DIST and ref_box[1] <
          comp_box[1]:
        changes = True
        ref_box[1] = comp_box[1]
        to_remove.append(j)
      else:
        pass
    new_boxes.append(ref_box)
  if changes:
    return mergeBoxes(new_boxes)
  else:
    return new_boxes


def removeSmallBoxes(boxes, image, xvals):
  new_boxes = []
  for i,box in enumerate(boxes):
    count = 0
    big_enough = False
    y = box[0]
    while y <= box[1]:
      for x in xvals:
        if image[y,x][0] == 0 and image[y,x
            ][1] == 255 and image[y,x][2] ==
             0:
          count = count + 1
      if count > MIN_CLUSTER_POINTS:
        big_enough = True
        break
      y = y + 1
    if big_enough:
      new_boxes.append(box)
  return new_boxes

MIN_LINE_LENGTH = 3
MAX_SLOPE = 5
CORE_WIDTH = 31
MAX_DISTANCE = 8
CORE1_XVALS = range(1,31)
CORE2_XVALS = range(32,62)
CORE3_XVALS = range(63,94)
NUM_CLUSTERS = 20
MIN_CLUSTER_POINTS = 8
SNAP_DIST = 9

removed_boxes = []
added_box_points = []
added_boxes = []
adding = False
##--------------Program----------------

# Copy to a new image so the original doesn
    't get alterred

image_path = sys.argv[1]
new_image_path = os.path.splitext(
    image_path)[0]+"_alterred"+os.path.
    splitext(image_path)[1]
sub.call(["cp", "-f", image_path,
    new_image_path])

# For later when we take user input
image_with_deleted_boxes = os.path.splitext
    (image_path)[0]+"_with_boxes_removed"+
    os.path.splitext(image_path)[1]
sub.call(["cp", "-f", image_path,
    image_with_deleted_boxes])
```

```
im_w_del = cv2.imread(
    image_with_deleted_boxes)

image_with_added_boxes = os.path.splitext(
    image_path)[0]+"_with_boxes_added"+os.
    path.splitext(image_path)[1]
sub.call(["cp", "-f", image_path,
    image_with_added_boxes])
im_w_add = cv2.imread(
    image_with_added_boxes)

# Read in new image for editing

image = cv2.imread(new_image_path)

# Apply original ELSDc with no modification
     to parameters

grey_image = cv2.cvtColor(image, cv2.
    COLOR_BGR2GRAY)
grey_image_path = os.path.splitext(
    new_image_path)[0]+".pgm"
cv2.imwrite(grey_image_path, grey_image)
home_dir = os.path.expanduser("~")
elsdc_path = home_dir + "/mthe493/catkin_ws
    /src/elsdc/src/elsdc"
sub.call([elsdc_path, grey_image_path])

# Convert output.svg to a txt file
    containing X,Y coordinates for all
    lines

svgToCoords(os.path.dirname(new_image_path)
    +"/output.svg")

# Draw all detected lines in a given colour

coords_file = os.path.dirname(
    new_image_path)+"/output_coordinates.
    txt"

cf = open(coords_file, 'r')

for i, line in enumerate(cf.readlines()):
  drawLine(line.rstrip("\n"), image)
  drawLine(line.rstrip("\n"), im_w_del)
  drawLine(line.rstrip("\n"), im_w_add)

cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',image)
cv2.waitKey(0)
cv2.destroyAllWindows
cf.close()

# Scan all 3 cores for areas of high
    density and 'box' them

core1_file = os.path.dirname(new_image_path
    )+"/output_core1.txt"
core2_file = os.path.dirname(new_image_path
    )+"/output_core2.txt"
core3_file = os.path.dirname(new_image_path
    )+"/output_core3.txt"

height = image.shape[0]

for i in [1, 2, 3]:
  if i == 1:
    f = open(core1_file, 'r')
    core1boxes = []
    xvals = CORE1_XVALS
  elif i == 2:
    f = open(core2_file, 'r')
    core2boxes = []
    xvals = CORE2_XVALS
  else:
    f = open(core3_file, 'r')
    core3boxes = []
    xvals = CORE3_XVALS

  data = []
  for line in f.readlines():
    line.rstrip(',\n')
    for point in line.split(","):
      if point != '\n':
        data.append((complex(point).real,
            complex(point).imag))

  (mu, clusters) = find_centers(data,
      NUM_CLUSTERS)
  colourClusters(mu, image)
  colourClusters(mu, im_w_del)
  colourClusters(mu, im_w_add)

  boxes = []

  for centroid in enumerate(mu):
    try:
      cluster_length = getClusterLength(
          clusters[centroid[0]])
      boxes.append([int(centroid[1][1]-int(
          cluster_length/2)), int(centroid
          [1][1]+int(cluster_length/2))])
    except KeyError:
      continue
  #box_file = os.path.dirname(
```

```
      new_image_path)+"/core" + str(i) + "
      _boxes.txt"
  if boxes != []:
    boxes = mergeBoxes(boxes)
    boxes = removeSmallBoxes(boxes,image,
        xvals)
    if i == 1:
      core1boxes = boxes
    elif i == 2:
      core2boxes = boxes
    else:
      core3boxes = boxes
    #saveBoxes(boxes, box_file)
    #colourBoxes(boxes, im_w_del, xvals, i)
    #colourBoxes(boxes, im_w_add, xvals, i)
    cv2.imwrite(new_image_path, image)
  f.close()

colourBoxes(core1boxes, image, CORE1_XVALS,
     1)
colourBoxes(core2boxes, image, CORE2_XVALS,
     2)
colourBoxes(core3boxes, image, CORE3_XVALS,
     3)
cv2.imwrite(new_image_path, image)

cv2.namedWindow('Click Any Boxes to Remove
    ', cv2.WINDOW_NORMAL)
cv2.setMouseCallback('Click Any Boxes to
    Remove', removeBox)
cv2.imshow('Click Any Boxes to Remove',
    image)
cv2.waitKey(0)
cv2.destroyAllWindows

core1_to_remove = []
core2_to_remove = []
core3_to_remove = []

for click in removed_boxes:
  if click[0] in CORE1_XVALS:
    for i,box in enumerate(core1boxes):
      if click[1] in range(box[0],box[1]):
        core1_to_remove.append(i)
  elif click[0] in CORE2_XVALS:
    for i,box in enumerate(core2boxes):
      if click[1] in range(box[0],box[1]):
        core2_to_remove.append(i)
  elif click[0] in CORE3_XVALS:
    for i,box in enumerate(core3boxes):
      if click[1] in range(box[0],box[1]):
        core3_to_remove.append(i)
  else:
    print "Click out of range, skipping this
          box deletion"

#print "CORE 1 TO REMOVE" + str(
    core1_to_remove)
core1_to_remove.sort(reverse=True)
for i in core1_to_remove:
  del core1boxes[i]

core2_to_remove.sort(reverse=True)
for i in core2_to_remove:
  del core2boxes[i]

core3_to_remove.sort(reverse=True)
for i in core3_to_remove:
  del core3boxes[i]

colourBoxes(core1boxes, im_w_del,
    CORE1_XVALS, 1)
colourBoxes(core2boxes, im_w_del,
    CORE2_XVALS, 2)
colourBoxes(core3boxes, im_w_del,
    CORE3_XVALS, 3)

cv2.imwrite(image_with_deleted_boxes,
    im_w_del)

cv2.namedWindow('Click and Drag to Create
    New Boxes', cv2.WINDOW_NORMAL)
cv2.setMouseCallback('Click and Drag to
    Create New Boxes', addBox)
cv2.imshow('Click and Drag to Create New
    Boxes', im_w_del)
cv2.waitKey(0)
cv2.destroyAllWindows

box = [(1,2),(3,4)]
for i,point in enumerate(added_box_points):
  if i%2 == 0:
    box[0] = point
  else:
    box[1] = point
    if box[0][1] > box[1][1]:
      tmp = box[0]
      box[0] = box[1]
      box[1] = tmp
    added_boxes.append(box)

for box in added_boxes:
  if box[0][0] in CORE1_XVALS and box[1][0]
       in CORE1_XVALS:
    core1boxes.append((box[0][1],box[1][1]))
  elif box[0][0] in CORE2_XVALS and box
      [1][0] in CORE2_XVALS:
    core2boxes.append((box[0][1],box[1][1]))
```

```
  elif box[0][0] in CORE3_XVALS and box
      [1][0] in CORE3_XVALS:
    core3boxes.append((box[0][1],box[1][1]))
  else:
    print "Box not included, xvals not in
        range"

colourBoxes(core1boxes, im_w_add,
    CORE1_XVALS, 1)
colourBoxes(core2boxes, im_w_add,
    CORE2_XVALS, 2)
colourBoxes(core3boxes, im_w_add,
    CORE3_XVALS, 3)

print len(core1boxes) + len(core2boxes) +
    len(core3boxes)

cv2.imwrite(image_with_added_boxes,
    im_w_add)

cv2.namedWindow('This image will be used to
     fit ellipses', cv2.WINDOW_NORMAL)
cv2.imshow('This image will be used to fit
    ellipses', im_w_add)
cv2.waitKey(0)
cv2.destroyAllWindows
```

```
imageFile = os.path.splitext(image_path)
    [0]+"_alterred.pgm"
global img
img = getImage(imageFile)
vLines = [0, 21, 33, 56, 70, 92]
targetFeatures = [ 1, 2, 3, 5, 6, 7, 8,
    9,10, 11, 12, 13, 19]
#targetFeatures = [ 1, 3, 5, 6, 8, 10, 11,
    12, 13, 19] # prez ones
#targetFeatures = [ 15, 16, 17, 18, 20] #
    bad ones
#targetFeatures = [6,19] # awesome example
    ones
display = 0 # 1 For output printing, 0 for
    none.


descTable = mother(coords_file, imageFile,
    vLines, targetFeatures, display)

for desc in descTable:
    print(desc)


plt.show()

#cv2.imwrite(new_image_path, image)
```

References

[1] S. Consulting, "Geotechnical Core Logging Back River Project," 2012.

[2] R. Viorica, Pierre, "A Parameterless Line Segment and Elliptical Arc Detector with Enhanced Ellipse Fitting," 2011.

[3] M. Randall, Grompone, "LSD: A fast line segment detector with a false detection control," 2010.

[4] J. Desolneux, Morel, "From Gestalt Theory to Image Analysis: A Probabilistic Approach," 2008.

[5] Haw-ren Fang, Yousef Saad, "Farthest Centroids Divisive Clustering ," 2008. [Online]. Available: http://www-users.cs.umn.edu/~saad/PDF/umsi-2008-05.pdf

[6] Wikipedia, "K-Means Clustering." [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering

[7] Jason Corso , "Clustering in Computer Vision." [Online]. Available: https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_1006_clustering.pdf

[8] J. Ran, "Learning From Ground Failures," 2007.

[9] Dave Petley, "Anargyroi: an 80 million cubic metre mining-induced landslide in Greece this weekend," 2017. [Online]. Available: https://blogs.agu.org/landslideblog/2017/06/12/anargyroi-1/

[10] Garima Sehgal, Dr. Kanwal Garg, "Comparison of Various Clustering Algorithms," 2014. [Online]. Available: http://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit2014050387.pdf

[11] Specim Ltd., "Hyperspectral imaging in geology," (n.d.). [Online]. Available: http://www.specim.fi/hyperspectral-imaging-in-geology

[12] Bruker, "X-ray diffraction ," (n.d.). [Online]. Available: https://www.bruker.com/products/x-ray-diffraction-and-elemental-analysis/x-ray-diffraction.html

[13] Z. Wang, "Automated extraction of building geometric features from raw LiDAR data," 2009.

[14] J. Kissi, "Automatic Fracture Orientation Extraction from SfM Point Clouds," 2016.

[15] P. Hough, " Method and means for recognizing complex patterns," 1962.

[16] Rosin, "Ellipse fitting using orthogonal hyperbolae and Stirlings oval," 1988.

[17] P. Boggs and J. Rogers, "Orthogonal Distance Regression," 1989.

[18] J. R. P. Boggs, R. Byrd and R. Schnabel, "Users Reference Guide for ODRPACK Version 2.01 Software for Weighted Orthogonal Distance Regression," 1992.

[19] M. Gulliksson and I. Sderkvist, "Surface fitting and parameter estimation with nonlinear least squares," 1995.

[20] R. B. P. Boggs and R. Schnabel, "A Stable and Efficient Algorithm for Nonlinear Orthogonal Distance Regression," 1987.

[21] J. Dennis and R. Schnabel, "Numerical Methods for Unconstrained Optimization and Nonlinear Equations," 1987.

[22] Statista, "Canadian Mining Industry ," (n.d.). [Online]. Available: https://www.statista.com/topics/3067/canada-s-mining-industry/

[23] Mining Association of Canada, "Mining Facts ," (n.d.). [Online]. Available: http://mining.ca/resources/mining-facts

[24] Mining.com, "Mining industry value drops below one trillion dollars ," (2015). [Online]. Available: http://www.mining.com/mining-industry-value-drops-below-one-trillion-dollars/

[25] Institut de le statistique Quebec, "Core Drilling ," (2016). [Online]. Available: http://www.stat.gouv.qc.ca/statistiques/mines/forage-carottier/index_an.html

[26] Datamine, "Core Profiler," (n.d.). [Online]. Available: https://www.coreprofiler.com/about

[27] Institut de la Statistique du Quebec. [Online]. Available: http://www.stat.gouv.qc.ca/statistiques/mines/mines-en-chiffres-2013_an.pdf

[28] K. Martyniuk, "Mobile LiDar scanning and processor for automated structural evaluation of discontinuities in drill cores," 2017.

[29] John Lisle Orpen, "Rock core logging," Patent WO2 007 102 129A2.

[30] C. L. C. of the South Africa, "A Guide to Core Logging for Rock Engineering," 1976.

[31] A. Sutskever, Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2014.

[32] J. Canny, "A computational approach to edge detection," 1986.

[33] K. Xu, Oja, " A new curve detection method: Randomized Hough Transform," 1990.

[34] G. C. K. J. Matas, J., " Progressive probabilistic hough transform," 1998.

[35] Azad Naik, "K-Means Clustering Algorithm." [Online]. Available: https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm

[36] W. C. Z. Szpak and A. van den Hengel, "A Comparison of Ellipse Fitting Methods and Implications for Multiple-View Geometry Estimation," 2012.

[37] D. Harrison, "Computational Physics Background Material for Expt. 1 - Fitting Techniques," (n.d.). [Online]. Available: https://faraday.physics.utoronto.ca/PVB/Harrison/Fitting.pdf

[38] R. B. P. Boggs and R. Schnabel, "A Stable and Efficient Algorithm for Nonlinear Orthogonal Distance Regression," 1985.

[39] G. G. W. Gander and R. Strebel, "Least-Squares Fitting of Circles and Ellipses," (n.d.).

[40] N. Redding, "Implicit Polynomials, Orthogonal Distance Regression, and the Closest Point on a Curve," 1995.

[41] G. Watson, "Some Problems in Orthogonal Distance and Non-Orthogonal Distance Regression," 2001.

[42] L. Button, "Large-Scale Machine Learning with Stochastic Gradient Descent," 2011.

[43] ——, "Stochastic Gradient Learning in Neural Networks ," n.d.

[44] M.N.S. Swam, "Search and Optimization by Metaheuristics,," 2016.

[45] Paul Coddington, "Simulated Annealing and Optimization," 1998.

[46] Frik Els, "Mining exploration spending drops to 11-year low," 2017. [Online]. Available: http://www.mining.com/greenfields-share-exploration-spending-drops-record-low/

[47] BullMarketRun, "How Big Discoveries Are Missed Sometimes By Just Tens Of Meters!" 2017. [Online]. Available: http://www.bullmarketrun.com/big-discoveries-missed-sometimes-just-tens-meters/

[48] MINING.com, "British Columbia mining revenue was up $1B last year," 2017. [Online]. Available: http://www.mining.com/web/net-mining-revenue-up-1b-in-bc-last-year/

[49] Cherise Seucharan, "Mount Polley mine disaster: 3 years later concerns still remain," 2017. [Online]. Available: http://www.cbc.ca/news/canada/british-columbia/mount-polley-mining-fears-1.4235913

[50] Joe Foy, "Cleaning Up BC's Dirty Mining Industry," 2016. [Online]. Available: https://clayoquotaction.org/wp-content/uploads/2016/06/2016_mining_paper_final_web.pdf

[51] Judith Lavoie, "Whos Paying for the Clean Up of the Worst Mining Spill in Canadian History?" 2017. [Online]. Available: https://thetyee.ca/News/2017/03/31/Who-Pays-for-Mount-Polley-Spill/

[52] DW, "German officials reveal cause of 2009 Cologne archive collapse," 2017. [Online]. Available: http://www.dw.com/en/german-officials-reveal-cause-of-2009-cologne-archive-collapse/a-38805218

[53] Kate Good, "5 Ways Coal Mining is Destroying the Planet," 2014. [Online]. Available: https://www.onegreenplanet.org/animalsandnature/5-ways-coal-mining-is-destroying-the-planet/

[54] John E. Young, "Mining the Earth," 1992. [Online]. Available: https://www.uow.edu.au/~sharonb/STS300/sustain/mining/impactarticle1.html

[55] The World Bank, "Waste Generation," 2010. [Online]. Available: http://siteresources.worldbank.org/INTURBANDEVELOPMENT/Resources/336387-1334852610766/Chap3.pdf

[56] K. Hudson-Edwards, M. Macklin, P. Byrne, G. Bird, and P. Brewer, "The environmental impact of the Mount Polley mine tailings spill and related clean-up operations, British Columbia, Canada," 2015.

[57] Todd Lohr, "The Ethics Of Digital Labor," 2017. [Online]. Available: https://www.forbes.com/sites/kpmg/2017/08/16/the-ethics-of-digital-labor/#33309bff4085

[58] KPMG, "An ethical compass in the automation age," 2017. [Online]. Available: http://www.kpmg-institutes.com/content/dam/kpmg/advisory-institute/pdf/2017/ethical-compass-automation-age.pdf

[59] D. Mashile-Nkosi, "Private Sector for Development: How Private Investments have helped Mining Sector Growth," 2014. [Online]. Available: http://ecdpm.org/great-insights/promoting-development-through-business/private-investments-mining-sector/