# CS/IT Honours
# Final Paper 2021

Title: Template Matching for Orchard Structure Classification

Author: Jonathon Everatt

Project Abbreviation: ORCHSC

Supervisor(s): Patrick Marais

| **Category** | ***Min*** | ***Max*** | **Chosen** |
|---|---|---|---|
| Requirement Analysis and Design | 0 | 20 | 5 |
| Theoretical Analysis | 0 | 25 | 0 |
| Experiment Design and Execution | 0 | 20 | 0 |
| System Development and Implementation | 0 | 20 | 20 |
| Results, Findings and Conclusions | 10 | 20 | 20 |
| Aim Formulation and Background Work | 10 | 15 | 15 |
| Quality of Paper Writing and Presentation | 10 | | 10 |
| Quality of Deliverables | 10 | | 10 |
| Overall General Project Evaluation (this section allowed only with motivation letter from supervisor) | 0 | 10 | 0 |
| **Total marks** | | **80** | **80** |

**Template Matching for Orchard Structure Classification**

Jonathon Everatt

Department of Computer Science

University of Cape Town

September 2021

**Abstract**

*This paper proposed a template matching based approach developed in python to determine the planting pattern present in a GeoJSON dataset of an orchard. A secondary aim is to extract key parameters about the orchard such as intra-row spacing, inter-row spacing and tree count. A 3-step algorithm will be implemented to achieve these aims. The algorithm will pre-process the data and extract parameters such as scale and rotation necessary for the template matching algorithm to execute. The output of the template matching algorithm is scored based on the number of matches and correlation level as well as how well the template matches the parameters of the data. The planting patterns that will be tested in this paper are Square, Rectangle, Hexagonal, Double Hedgerow, and Triangle. These are tested on idealised data and real-world datasets of orchards. The results will be analyzed to see how well template matching algorithms could detect that pattern present and if key parameters were correctly extracted.*

**CCS Concepts**

Computer Vision, Pattern Classification, KD Trees, Template Matching, Planting Patterns

**Keywords**

Template Matching, KD Trees, Nearest Neighbour, Normalized Cross-Correlation Image Processing, Image Analysis

**1. Introduction**

Crop management is an integral task in farming. Improving management can lead to faster problem discovery and optimisation of the farm. Currently, orchard inspection is carried out by workers on the plantation. However, with the advancement of unmanned aerial vehicles (UAVs), it is possible to get a bird's eye view of the orchard. Automation of this process can provide the farmers with better information for farm management which leads to increased production [1]. The information that this project aims to ascertain is the planting pattern present and key parameters about the orchard.

Planting patterns are necessary to orchards for optimising the growth and yield of their crops. There are multiple types of planting patterns as different crops require different amounts of spacing between them to achieve optimal growth (ie. the density balance) [2]. These patterns can have different effects on different plant species ranging from a difference in yield to maximum plant growth and even ease of maintenance [3]. For reasons such as these, it is important to constantly monitor the planting patterns of orchards. And the patterns used in a farm may vary depending on the species and soil type present. The patterns that will be tested in this paper are Square, Rectangle, Hexagonal, Double Pair, Hedge Row, and Triangle.

This paper proposes a template matching based approach for detecting patterns in a GeoJSON dataset of an orchard. Template matching is the process of determining the presence and possible position of a smaller image, called a template, in a larger image [4]. Template matching has already been successfully implemented for tree detection from a bird's eye view using the GeoJSON aerial data that this system will use [5]. Planting patterns are made up of repeating geometric shapes (Appendix A). This makes template matching an ideal candidate for the detection of these patterns as it can use geometric shape templates to determine the pattern present. If a specific shape has been repeated enough in the orchard that can determine the planting pattern that is present. Patterns can be determined based on the number of matches and the levels of correlation between the templates and the source image that is paired with each match.

For the Template Matching algorithm to work on the data and determine the correct pattern, a 3-step algorithm will be used. The first step is pre-processing and parameter extraction. This step uses KD Trees, trigonometric functions and row detection to extract the parameters of the planting pattern [6]. Template matching has two parameters namely scale and rotation that need to be solved as well. The image will be orientated according to the rotation with the rows being aligned with the y-axis and the templates generated according to detected scale(s). The scale and rotation information from step 1 is used for step 2, the Template Matching Algorithm, to run and detect matches from generated templates of each of the planting patterns. The correlation levels and the number of matches are sent to step 3, Analysis of matches, which will determine the patterns which could be present in the data. This is done by scoring the results of step 2 based on the number of images, correlation levels and how well the template adheres to extracted parameters. This step is necessary due to the similarity that exists between some of the patterns for which the system will test.

This project's primary aim is to detect the planting pattern in a GeoJSON dataset using a Template Matching Algorithm. A complete success case would be the definitive determination of a specific pattern, but due to similarities that exist between patterns, a partial success case would give several likely patterns including the correct one. The secondary aim of the project is parameter extraction. The parameters are key information values about the dataset given to the program. These include, but are not limited to, intra-row spacing,

inter-row spacing and the position of corner trees. While parameter extraction is a secondary aim of the project, some of the values such as the spacings are required to be found for the template matching algorithm. Success in this aim would be either partial or (ideally) full extraction of parameters. It is acceptable if a parameter cannot be detected, but not if it is incorrectly detected. Partial success is if only some of the parameters are extracted correctly and others incorrectly but these are consistent over multiple experiments. And a failure for this aim would constitute if over multiple experiments the parameters were incorrect with no reasonable cause. A reasonable cause would constitute extreme levels of noise, curved rows or multiple planting patterns.

The paper is organised in the following way: Section 2 summarises Template Matching and the implementation that will be used. Section 3 describes the design of the system and how it is intended to achieve the aims, as well as why it was developed in that order. Section 4 discusses the results of the system when tested on ground truth data and real-world GeoJSON data. Section 5 and 6 concludes the paper and discusses possible future work.

**2. Background and Related Works**

Template matching is the process of determining whether predetermined sub-image(s) are present within a larger source image(s) and the approximate location of those templates [4]. Template matching techniques have a wide range of applications in Computer Vision, from object detection to feature tracking. Due to their prevalence in Computer Vision and being a commonly used Computer Vision technique, there is a wide scope of knowledge and research about them [5].

Template matching techniques have been used for similar data types before. Zainuddin and Daliman [7] used template matching to successfully identify tree species from aerial footage. Template matching has also been used to count trees present in aerial footage [8]. This shows template matching is compatible with aerial footage and can be used to detect patterns.

There are 3 categories of template matching techniques: intensity-based, feature-based, and Relational. For this project, we decided to use intensity-based techniques because they were the best fit for repeating geometric shapes [9]. Intensity-based techniques test the pixels of a template against the pixel using an equation called a similarity measure [10]. Several types of similarity measure output correlation value and save it into an image matrix[11]. This can be used to determine the correlation at each position of the template on the image.

Examples of similarity measures that are used by Intensity-based techniques are NCC (normalized cross-correlation), SAD (Sum of Absolute Difference), and SSD (Sum of Squared Difference) [12]. The similarity measure that is used for this project is NCC. This was chosen because it is more robust in measuring similarity between the template and the image [13] and can be characterized by the equation below.

$$NCC(\gamma) = \frac{\sum(f(x,y)-f'u,v)(t(x-u,y-v)-t')x,y}{\sqrt{\sum(f(x,y)-f`(u,v))^2 x,y \sum(t(x-u,y-v)-t`)^2 x,y}}$$

equation 1: NCC similarity equation[14]

NCC makes use of multiplication more times than the other similarity measures, so it is relatively more computationally expensive, but the difference is negligible with the computational power of modern GPUs [14, 15]. So, it is more valuable to have a robust measure for the noise present in the datasets than then lighten the computational load. The system will return the position matches in the image matrix that has a correlation score above a certain threshold (0.99 is a perfect match, 0.00 is a complete mismatch)[14]. This is because the Template Matching algorithm returns image matrices of the correlation match at each position of the template while it is being slid over the images [10].

OpenCV has an implementation of template matching in python that is used for the project. This implementation has intensity-based algorithms and can select which similarity measure to use (such as NCC, SAD, SSD) [18]. The OpenCV library has been found to have expensive computational requirements on larger images. So it is necessary to resize them beforehand to optimize efficiency [17].

Intensity-based template matching is efficient and easy to implement. The problem with it is that it cannot inherently deal with testing multiple rotations and scales [11]. Template of a certain scale will not return a high correlation match or even a match for the same template of a different scale [19]. The only way a pure template matching algorithm can deal with this problem is through brute force. But trying to test for every possible scale will create a massive computational load and is inefficient [9]. Determining an exact scale is complicated and can lead to incorrect results of correlation and scale when dealing with noise [20].

The other problem that template matching techniques face is the problem of rotation (As shown in the background literature) [21]. While feature-based or relational techniques can account for scales using techniques like Best-Buddies Pair. Intensity-based techniques cannot account for these problems [13, 22]. A naive implementation also suffers the same problem that scale faces, namely an extreme amount of unnecessary computational load [20]. So it is necessary to determine the scale or a small range of values that contain the scale. Or to orient all images with the same rotation so that you can eliminate the problem of rotation [20].

Template matching does calculations based on each pixel in a template and image. These calculations are efficient because they can be done independently and are highly parallelizable [23]. Thus by having and implementing Template Matching Techniques on a GPU you can achieve a speed-up of up to 16x [21]. This allows for leeway in detecting scale and rotation parameters. The template matching algorithm is not computationally expensive to the degree where it is necessary to determine exact numbers but a small range containing the true value(s) is sufficient [23].

## 3. Design and Implementation

As can be seen in figure 1 the system has 3 major steps which it undergoes. The input taken by the system is GeoJSON files with polygon coordinates of the plants stored in World Geodetic System 1984 format and the output is in a text file format with a list of the detected parameters and the detected patterns (along with the scores for all patterns). The system was developed in python due to its large number of available libraries such as OpenCV, matplotlib, shapely and numpy that were used in the project.

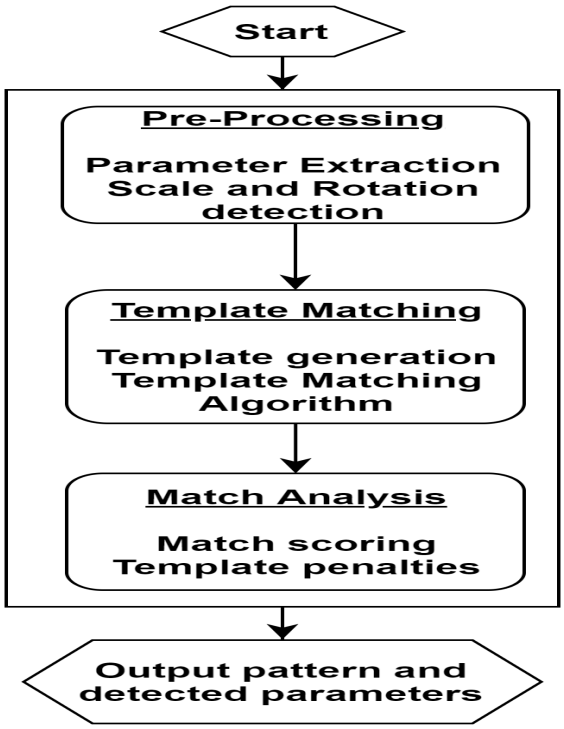



*Figure 1: System Pipeline*

The system works on certain assumptions about the data. The first assumption is that the rows are straight. It also assumes that all the rows are roughly parallel to each other so there is only 1 rotation needed to orient the rows onto the y-axis[1]. The row detection and rotation calculation will return an incorrect result for significantly curved rows and this will result in the Template Matching algorithm failing.

The second assumption is that there is only one planting pattern present in the data. If there is more than one the system will not give a definitive answer about which pattern is present. A graph can be displayed or saved which shows the location of the matches for the planting pattern template that was used. Finally, certain assumptions are made about the orchards: intra-row spacing and inter-row spacing is consistent among the majority of the data, so a mean can be representative of the spacing in the dataset. And the planting patterns are ones that templates are made to detect. Also that there is no significant difference of elevation in the dataset

[1] This is not true for all of the real-world data as will be seen in the results section. Thus ways to overcome the limitation will be discussed there or only a single rotation will be dealt with at a time.

### 3.1 Pre-processing

The pre-processing step extracts the information about the dataset necessary for the system. There are two aspects of information to be extracted: Template Matching problem solving and Parameter extraction. The GeoJSON files details polygons for which the program calculates a centroid. The problem is the amount of noise in the polygon centroid. There are two reasons for this noise: firstly, it is real world data so there are numerous factors that can lead to imperfect spacing in the planting phase. The second reason is because we are taking a centroid of a polygon that is roughly a circle, but because they aren't perfect circles there is some drift off its place on the row from that. For the calculations to determine answers for each of the aspects of this step, the data is put in a KD Tree. This allows us to efficiently look up N nearest-neighbours and the distances which are important for parameter extraction. By finding the nearest neighbours and considering the relative angles between points we can detect straight rows in the data. This enables the program to determine the rotation of the image and further extract more parameters.

**Row Detection**

To extract some number of trees per row from the dataset and determine the rotation, a row detected algorithm was used. By detecting a row one can determine the angle of the row, and use that to orient the image so the rows are on the y-axis. The algorithm runs until it has detected 10 rows with more than 10 points or there more points have been detected than in the dataset. This is because due to noise and missing data points row detection can fail or incorrectly detect the rows but 10 rows of 10 points tested as the best balance of efficiency and robustness. Also anything greater than 10 points becomes very difficult for the algorithm to detect enough rows.

Row Detection Algorithm

```
1.  Initialise list KD Tree (4 levels) and lists
2.  Pick a random starting point
3.  While in a row:
4.    For each level in KD Tree
5.      If angle from current point to point[level]
    is in range of row angle
6.       Add point to row list
7.       Break //for loop
8.       Else: if end of KD Tree end while loop
9.  Repeat from origin with inverse average angle
10. Return row list
```

To calculate the angle of the line between two points we use equation 2.

$$\sigma = atan2(y2 - y1, x2 - x1)$$

*Equation 2: Line rotation*

When checking if the angle from current point to possible next is in range. The algorithm checks to see if the angle from the current point to the next and the origin to the next are within reasonable bounds (30 degrees for current and 10 degrees for origin). The origin angle is used for the midpoint levels of the row as the points get further points in other rows that satisfy that clause. So it needs to be combined with the angle from the current point to make sure the algorithm stays within the same row. An angle is range is calculated off the threshold values where:

*If abs(angle1-angle) < threshold: True else: False*

The row detection algorithm tends to complete a row either at the end of the row or when it encounters significant noise. Such as a large gap in the row, if a point is far away from the

row or if there is a lot of noise present that alters the average angle off the row.

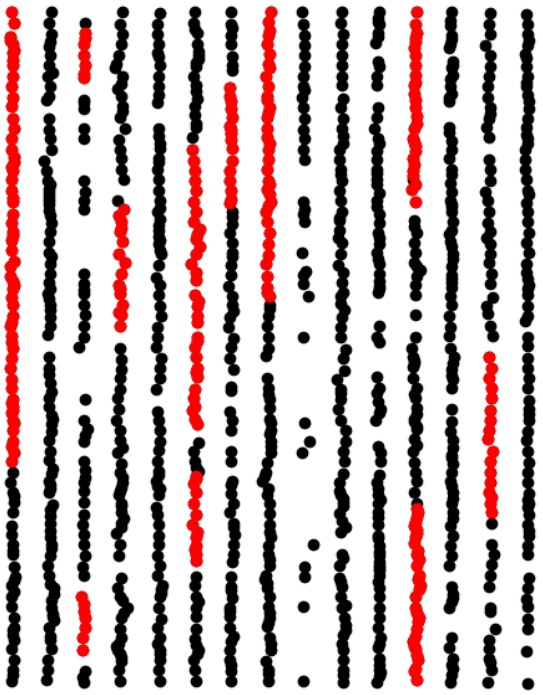


*Image 1: Row Detection. Red is detected points*

The above image shows the points picked up by the row detection (the red points) algorithm executed on dataset 43581. It is not a perfect detection algorithm as the noise in real-world data can end the row prematurely. But it does sufficiently find enough of the row to determine the rotation of the rows (provided the assumption of straight rows is met).

**Template Matching Problem Solving**

***Rotation***

Once 10 rows of sufficient length are detected by the row detection algorithm the relative angles are then analysed to determine which rotation is a good approximation for the image. The angle of a row is calculated by averaging the angle of lines between consecutive points using equation 2 (see the previous section)

The averaged angles are then analyzed to determine which angle range is most common (e.g. 80 <= x <= 100), in 20-degree increments. The average angle of the most common range is then used for the rotation of the image and is increased by 90 degrees to align the rows onto the y-axis. The reason we use common ranges is that the square pattern would return perpendicular row lengths. This has the side effect of being robust against multiple rotations being present in the image as well (side effect because 1 rotation is the assumption). In addition, the specific rotations of a row could have a difference of 180 degrees depending on which direction the algorithm started with. If there is a tie with the most common angle range, the difference between the two angles is calculated. If the difference is close to 180 degrees that means that they are different angles of the same line so the greater option is chosen because this is always positive. If the difference is not close to 180 degrees the program will output a warning to the user that more than one rotation was detected and choose the greater detection to use in its execution. The results are also warned to possibly be inaccurate. Even if there are two or more rotations present, the random sampling of origin points for the row detection algorithm will likely end up sampling a specific rotation more often than the others.

***Scale***

There are two aspects to the scale issue. The first is that the number of points dealt with by the system can be too large for an image to reliably display them, and have enough spacing in between to differentiate between patterns. This is solved by splitting up each point set into sub-images, that contain around 500 points each, on which the template matching is run. The sub-images are graphs generated by matplotlib.pyplot. With the axes set to be equal to preserve the patterns. The axes are also hidden from the graph and any pixel value greater than 0 is set to 255. This results in points that look like:

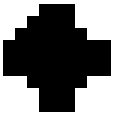


*Image 2: Template and sub-image point*

The sub-images are aligned with the detected rotation because the row detection algorithm cannot process them individually as it takes Point Set input and not an image. Sub-images also promote modular design for the system as larger datasets will still be dealing with similar images but a greater quantity of them. The OpenCV template matching algorithm also loses efficiency with an increase of image size so this optimises the efficiency of the template matching algorithm the system uses (See OpenCV section in background literature) [17].

The second problem is the determination of scale for the template between the points. The pixel difference can be associated with two parameters that are extracted: intra-row and inter-row spacing. The images in this case are orientated with rows on the y axis. This makes it possible to determine the number of pixels between each point and store them in a dictionary. This means that you can look up the difference value each time it is calculated and increase the counter on it. The difference is determined by iterating through the image and finding the number of white pixels between every detected black pixel. Eventually, the distance values greater than 75% of the max is added to the list of values of scales. This algorithm is dependent on correct rotation detection.

It is necessary to note that for certain patterns (e.g. Triangle, Hexagonal) a direct line on the x-axis would skip rows and return double the length. So for these patterns, the value is halved, so long as it is sufficiently large. The average pixel distance for a 500 point image is around 20 to 60 pixels with this image generation method depending on the pattern. Anything greater than 20is considered sufficiently large to halve. This is because any image with a spacing of less than 10 would cause the template to generate incorrectly and the program would fail anyway. The way templates are generated requires a distance of 10 pixels because each point in the template and sub-image are generated as a black '+' that is 9 pixels in diameter. When generating the templates it is necessary to maintain a minimum distance of 10 to accurately represent the pixels and have sufficient space to differentiate, even on an oversampled level, the relative spacing between the points. 10 pixels is the minimum value that can differentiate reasonable differences in correlation matches between the templates.

**Parameter Extraction**
The secondary goal of the system is to extract parameters about the dataset. Parameters that can be extracted are: total detected tree count, intra-row spacing, inter-row spacing, number of rows, number of trees per row, and position of corner trees. These parameters are extracted in the pre-processing step. The Tree Count is the simplest to detect as it's the length of the  Point Set imported into the program.

Using KD Trees for nearest neighbour lookups and distance values the system can determine intra-row and inter-row spacing. The closest point on every planting pattern should always be within the same row (except square but then it doesn't matter). By taking the mean distance of the nearest neighbour for each point in the KD Tree, you get the mean of the intra-row spacing. For the inter-row spacing, you have the closest point that is roughly perpendicular or roughly 60 degrees to the angle of the line of intra-row spacing, so you know the point is in a different row. The angle can either be perpendicular or roughly 60 degrees to accommodate patterns based on quadrilateral or triangles. Then by averaging these values you can get the inter-row spacing. There is some minor variance here, depending on how close the angle is to perpendicular but this is acceptable for the system.

The row parameters such as average number per row, number of rows and presence of a road or ditch. The row detection logic can be repurposed into row counting.

Row Counting Algorithm

```
1.  Angle = (Row Angle -90) % 180
2.  Dataset = KDTree(PointSet)
3.  Nearest_dist, nearest_ind = data.query(k=10)
4.  Point_list = []
5.  point_list.append(random point in row)
6.  Flag = True
7.  While Flag
8.      For x in range(10)
9.          Point = nearest_ind[point_list[-1]][x]
10.         If angle(Point_list[-1] and Point) close to Angle
11.         Point_list.append(point)
12. Return Point_list
```

The pseudocode above shows how the system determines the number of rows. It needs to be run twice to determine count rows going in both directions from the origin point. The distances can also be saved in an array if there are outlier distances that show the presence of roads or ditches. The average of the other (non-outlier) distances make up the inter-row spacing for the dataset. To count the average row length Take a list of the number of points in the row and calculate the average of every value that is greater than 0.6 * Max value. This is to counter any noise or breaks in a row. This helps the answers not be weighted by rows that are not fully detected due to noise. For example, if there is a significant break in the middle of the row then the row detection would determine it is at the end of the row.

To find the coordinates of a corner tree, the system samples 4 possible locations. The points that have the minimum and maximum values of x and y in the Point Set. Since this is calculated from the Point Set, it is unaffected by the rotation step above. This solution is rather simple but can be prone to incorrect results in certain cases. If the image is orientated close to 0 or 90 degrees then noise could confound the results and could return a point somewhere off-centre in the row. Parameter extraction is a secondary aim and extra effort to solve this perfectly is computationally and time-wise not worth it. This approximation of the corner trees is always good enough because although it doesn't get the exact trees, they are normally within 3 or 4 trees of the true corner.

**3.2 Template Matching**
The system uses the template matching algorithm from the OpenCV (cv2) library and makes use of a Normalized Cross-Correlation similarity measure (see background section). The same library is also used to draw rectangles around the matched points. These rectangles can be used to display visual output to the user to allow them to see where there are or aren't matches. The source image used in the template matching algorithm is a white image with the coordinates of every plant being a black '+' sign (see image 2) with the lines having a breadth of 3 pixels. The same coordinate points are on the templates allowing for a very high correlation match (>0.8). Correlation matches greater than 0.4 are still considered for the match analyzer but are not of significant value. Anything below a correlation of 0.4 can be attributed as a false match because at this point noise would become a significant issue with creating false matches. The high correlation matches will have the most weight in the decision of planting pattern with each correlation is essentially worth half the level before it (0.1 decrements). The templates are generated at every scale but the Template Matching Algorithm but the best fit template is the only template considered per pattern.

**Template Generation**
Five template types are generated to match five of the most common planting patterns: Square, Rectangle, Hexagonal, Double Hedge Row and Triangle. For examples of the templates with their respective patterns see Appendix A. The template for Hexagonal is an equilateral triangle that is repeated six times in the pattern to form a hexagon (*See fig 2*). The triangle template uses Isosceles triangles. Three templates have two different scale parameters: Rectangle, double hedgerow and triangle. For the sake of matching it is required that these templates have different enough values to differentiate from other templates like the square template. Thus the longer side is required to be at least five pixels larger than the shorter side.

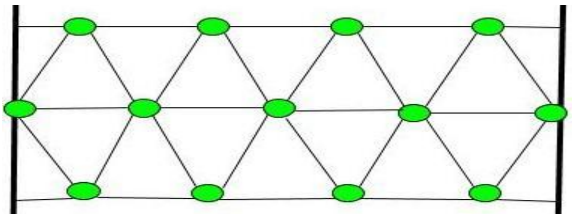


*Figure 2:*

*Hexagonal pattern made up of 6 equilateral triangles*

The templates are generated off the detected scales and are all the same size. This is because a comparison of templates with different sizes produces incorrect results. The most common reason is a triangle template matches 2 points onto a rectangle or square pattern and mismatches one point. This gives a moderate correlation and scores higher than the true pattern. So it is necessary for a constant template size to avoid this problem.

**3.3 Analysis of Matches**
The matches are analysed to see which pattern is detected in the dataset. Each pattern has a list of the number of matches found above each correlation from 0.9 to 0.4. The patterns are scored based on the values in the list. Then the score is penalised against the spacing parameters that were detected in the program.

For the scoring of the matches, each correlation level is worth double the points of the level below it. With a correlation above 0.9 being worth 32 times as much as a correlation between 0.4 and 0.5. This is further reinforced by the fact that a dataset with very high correlations will also have a lot of lower correlations around the same point as the template is displaced from the high correlation point more and more. These factors work together and patterns that we want, with high correlation values, will score higher than patterns that have false positive matches at a low correlation value (around 0.4). In practice, a single point of a 0.9 match will be worth more than over 100 0.4 which is accurate and in line with the aim of the program.

When dealing with real-world data, high correlation matches are rare (See Appendix B for examples). Thus it is necessary to penalise certain pattern scoring to prevent template similarities from giving false positives (see Quincunx problem in the results section for an explanation of template similarity problem). Templates are penalised on two accounts, angle and scale. The scale penalisation happens when the scales are too close together to create a rectangle, double hedgerow or isosceles triangle template. Or when they are too different to create an equilateral triangle or square template. In these cases, the scores are halved, which effectively eliminates these templates from contention. When the values are too close if they cover very similar and small ranges. If they share a range and the maximum value is less than 1.5 times the minimum value templates with different side lengths are penalised, and vice-versa for if the maximum value is greater. If the range of both is overlapping but there is a big range, no penalty is applied.

The angle penalisation is more prone to noise so it is less harsh. If the perpendicular angle is closer to 60 degrees on average than 90 degrees; square, rectangle and double hedgerow are penalised. The penalisation, in this case, is a 25% reduction in the scores of the penalised patterns. The penalty also works against the triangle templates if the value is closer to 90 degrees. This angle is calculated during the inter-row spacing calculations and adds very little computational load onto the program. In real-world data noise is a problem so if the angle is indeterminate, between 68 and 82 degrees, there is no penalty applied.

For the final output of the score and pattern detection. The highest score is the most likely pattern detected. Patterns that have scores of greater than 75% of the high score are listed as possible detections by the program. This is generally due to too much noise being present to determine exactly which pattern is detected. The scores and number of matches at each correlation level for each pattern is also added to the output file so that the user can analyze the results.

## 4. Results and Discussion

To test the efficacy of Template Matching Techniques on the data 2 types of data are used. First is Ideal data with minimal noise that has a similar number of points to GeoJSON files. This is to show proof of concept and set ground truth data for the other experiments. The second is real-world data that uses the centroid of polygons from GeoJSON files. This data will see where the system succeeds and fails in a real-world scenario. The experiment questions are: Is a pattern correctly detected? Are the parameters correctly extracted? How is the performance/runtime? Why does the system succeed or fail? If failure, what are the reasons for failure/shortcomings?

### 4.1 The Quincunx Problem (Template Similarity Problem)

While conducting the tests of the system we identified an issue with a template for the Quincunx pattern that had to be removed from the system. The quincunx pattern is a square pattern with a centroid and when run through the row detection algorithm it is always rotated 45 degrees so the quincunx template is set as a '+'.

*Image 3: Quincunx pattern and template*

The interesting thing about this pattern is that it matches with a massive amount of false positives at a mid correlation level (0.4-0.7) for all the patterns that contain perpendicular angles. This led to it always being present in the detected patterns of Rectangle, square and double hedgerow and often outscoring those patterns.

Although the problem was a significant issue for the quincunx template, the double hedgerow template faces the same issue to a lesser extent. This is because if the spacings are not differentiated properly they will match across rows. This also highlights why the correct orientation is important for the templates. Because the double hedgerow were to match along a single row, rather than across many, it would become as indeterminable as the quincunx template. However, when matching into the double hedgerow dataset the square and rectangle templates have the same issue of matching equally as high. Although due to the difference in sides only one of the patterns will be detected, generally the rectangle.

The problem here is one of template similarity. The quincunx has features that are similar enough to each of the other previously listed templates so with the inclusion of noise it performs equally well or better than them even if not present. This is a drawback of the intensity-based template matching method as it struggles to differentiate with the details in images as it deals with them as a whole. It is significantly easier for a template matching algorithm to differentiate in patterns with large differences but when the differences rely on smaller details they can often get confounded with noise. This problem applies to a lesser extent with similarities between square/rectangle/double hedgerow and equilateral/isosceles triangles. The thing that separates these issues from the quincunx is that through the scoring system we can eliminate the issues. But the quincunx pattern can only be differentiated from the square pattern if you know the length of the sides so you can determine if you are measuring the distance between two corners or a corner and a centroid.

This can be extended to the other templates as well. There is kind of a Sorites Paradox with the geometric shapes, at what point does a square become a rectangle, when does an equilateral triangle become an isosceles triangle. A Sorites Paradox originates from vague and unclear statements with no clear line divide. Such as, how many grains of sand make a heap?[2] Classical geometry says a square has a perfectly equal side but that means in the real-world a square doesn't exist which does not make sense because there are things we consider as squares in everyday life, so we face the problem of

[2] A explanation of the paradox: https://plato.stanford.edu/entries/sorites-paradox/

when a square becomes a rectangle in a real world context. The Template similarity problem requires an answer to this paradox but a definitive, exact measure cannot be given. So the line is essentially arbitrary, the scoring system sets the line as >1.5x the length of the shorter side to be a rectangle. A persuasive argument could be made for any value between 1 and 1.5 or even a little more. It doesn't necessarily matter where the line is drawn, it matters more that there is a line for the system to work correctly.

**4.2 Ideal Data Experiment**

| Dataset | Pattern Detected | Parameters Extracted | Level of noise | Runtime | Number of Points |
|---|---|---|---|---|---|
| Square Ideal | Square | Yes | Low | 7.58 | 2500 |
| Triangle Ideal | Triangle | Yes | Low | 6.42 | 1250 |
| Rectangle Ideal | Rectangle | Yes | Low | 6.47 | 1250 |
| Hexagonal Ideal | Hexagonal | Yes | Low | 7.67 | 2500 |
| Double Hedge Row Ideal | Double HedgeRow Rectangle | Yes | Low | 10.78 | 4200 |

**Runtime**
The runtime for the idealized experiments are misleading for an 'in the wild' implementation of the system for two reasons. One is because the data is being imported from a text file which holds the centroid coordinate and not from a GeoJSON file that would need to calculate the centroid of the polygon. So although the number of points are comparable the amount of computational effort required to these points is around 20x less with idealized data (below on GeoJSON polygons are made up of at least 50 line strings).

The second reason is the noise of the data. Idealized data has minimal noise which reduces the amount of scales detected. This has a knock on effect of generating less templates and speeding up the system as a whole. As the noise in the data increases so does the runtime due to increased need for testing different templates.

**Parameter extraction**
The ability of the parameter will be properly tested when it comes to the real world data experiment. Although we can already see problems when it comes to the row count and number of trees per row. Both failed in a case to detect the correct parameter. However, we see in the other cases that the parameter was extracted. So there is some proof that the concept works. The spacing parameters are correctly detected and close to the true mean. The triangle dataset detected an inter-row spacing of 22.43m which is greater than the true mean. But that is likely due to the angle of the line between rows being 60 degrees and thus results in a longer line than a perpendicular angle.

**Pattern detection**
All of the patterns were correct on the first execution except double hedgerow . This is due to the template size issue. The size of the double hedgerow template was overlapping multiple points in the row and not only one. Where the other templates were generated to overlap just over two points. This was unfairly penalising the double hedgerow template. But, even by fixing the template size the square and rectangle templates scored well enough to be detected in the output of the Double Hedge Row data. The square template is only not detected by the system because of the penalty on its length. This is because of the Quincunx problem described above. Fortunately it only affects this one dataset and doesn't necessitate the removal of patterns. This is the main drawback of the template matching approach, the issue of template similarity.

Successful experiments on idealised data sets ground truth data for our system and all of the tests on idealised data successfully detected the pattern that was present in the data. This shows that in an ideal situation it is possible for the template matching to correctly identify the patterns in the data. The major difference between idealised data and real-world data is the amount of noise present. The next set of experiments will test how robust to noise the system is. Even if the next set of experiments are failures it just means that the system will have to adjust to deal with noise better, either using noise reduction techniques or possibly integrating feature-based template matching.

The scoring system was also very useful in avoiding the problem in the next section, the issue of template similarities. The problem is explained in detail in the next section but, essentially the scoring system successfully managed to draw a line between the similar templates and force the system to decide between one or another. The line is arguably arbitrary but there is no way to definitively choose and an arbitrary line that works is better than no line at all (as a definitive line does not exist).

**4.3 Real-World Data Experiment**
*For graphical examples of datasets see Appendix B*

| Dataset | Pattern Detected | Parameters Extracted | Level of noise | Runtime average | Number of Points |
|---|---|---|---|---|---|
| 43581 | Correct-Rectangle | Partially Correct | Low | 101s | 4992 |
| 36507 | Correct-Rectangle | Partially Correct | Low | 39s | 2412 |
| 36513 | Correct - Rectangle | Partially Correct | Moderate | 30s | 2322 |
| 35516 | Acceptable | Incorrect | High | 70s | 4280 |
| 41630 | No correct solution | Incorrect | High | 72s | 2891 |

**43581 Appendix B4**
*Results in Appendix C1*

43581 is a good baseline dataset for dealing with real-world data. There is a lot of noise that the system has to contend with containing drift and missing points. But not extreme to the point that the system is unable to deal with it. It also falls under the assumption of a single rotation and straight rows in the pattern. The rotation was identified well enough as we can see in Image 3 below. Furthermore, you can see that although the rectangle template could not match on all of the data it can successfully match on the parts with less noise. Introducing noise reduction techniques such as realigning the points back onto the rows would only improve the results of the system.

When it comes to parameter extraction the intra-row and inter-row spacing seem to be correctly identified with the distance being reasonable for an orchard. The fact that the inter-row spacing is greater than the intra-row spacing lends credence to a rectangle being the correct planting pattern identified. The image below shows all matches above a correlation of 0.4, with thicker rectangles showing higher match areas.

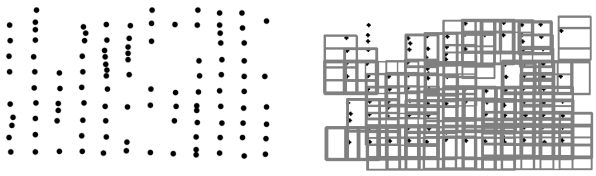


*Image 4: Unmatched and Matched sub-image*

The experiment was run 5 times to see where there is and isn't variance. The row detection was stable and detected the same angle on each execution. Which resulted in the template matching scoring returning the same scoring for each of the five experiments. Showing evidence that template matching is a real-world scenario is a viable solution. The runtimes were also consistent at around 100 seconds which shows that the program is stable and the random sampling methods do not make the runtime fluctuate wildly.

For the parameter extraction the intra-row spacing, corner tree detection, tree count and roads detected were constant. This is expected as they aren't reliant on random sampling. Intra-row spacing and tree count were also correct, while corner tree detection got a result that was close to the true corner but not quite. The dataset though was already angled with the rows 1 degree off the y-axis so this was an expected outcome (See the methodology section for why).

Row and road counting was completely incorrect and this is probably due to the amount of space between the rows. Using KD Trees for this situation is not the best solution, even trees with 10 levels may have too many points close to them in a row to detect a point across a row. The inter-row spacing was variable but most of the points were around 11 which is close to the true mean. So although there is variance in the answer it is a good enough approximation of the true mean to act as a solution for a secondary aim. Finally, the mean value per row fluctuated wildly. This is due to the row detection, it is expected that when dealing with real-world data this would fail. The solution for this issue is to use advanced row detecting algorithms that can properly deal with real-world levels of noise.

**36507 Appendix B1**
*Results in Appendix C2*

This dataset has the least amount of noise of the datasets available, but it does have a rotation that needs to be (and was) correctly identified. This dataset is the first real test for the row detection and rotation solutions and it was successfully detected. This shows that so long as we follow the straight-line assumption for a row this program will work. The system detected the rectangle pattern, which is correct.

The fact that the correct planting patterns are detected means that we correctly solved the rotation and scale issues that face template matching. We can also see in the scoring section that even without the penalty applied to the square and triangles (spacing and angle penalties) the rectangle pattern still would have been selected by the system as the representative pattern. The equilateral triangle had some high correlation matches due to noise (Figure C). But the system is robust enough to handle the noise and even without the penalty applied to the triangle the Rectangle template would be the only possible detected pattern.

| | |
|---|---|
| Square | 43343.75 [0, 0, 0, 28, 176, 923, 3093] |
| Rectangle | 345000.0 [0, 0, 20, 250, 976, 2408, 4983] |
| Isos | 25937.5 [0, 0, 0, 0, 38, 339, 1456] |
| Equi | 73000.0 [0, 0, 4, 27, 165, 698, 2362] |
| Dbl | 11312.5 [0, 0, 0, 0, 12, 157, 914] |

*Figure C: Scoring*

The row count was variable between 13 and 17. But this is a partial success as 16 to 24 are valid counts based on the algorithm. With the true answer being 24. Although the road detection was technically correct on 0, this is more likely a coincidence as it never detects roads. The mean value per row is also a very good approximation of the mean length of full rows present in the dataset. This is due to the row detection finding full rows in the dataset as we can see in the image below.

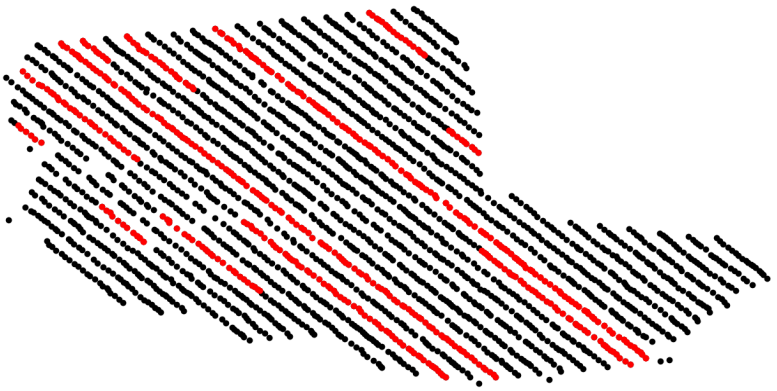


*Image 5: 36507 Row Detection Example*

**36513 Appendix B2**
*Results in Appendix C3*

*Image 6: Dataset 36513*

This dataset breaks one of the assumptions made about the data. That is that there are 2 clusters of plants present in the dataset with different rotations. The rows were very straight and there was minimal noise in them. Because the dataset broke the assumption we set the row detection rotation to prioritise the detection of the larger cluster (at the bottom of the image) which it normally prioritised anyway due to the difference in number of rows. The row detection still calculated its own rotation and was able to correctly detect it for the bottom cluster. The upper cluster was off rotation but all of the templates matched equally badly to that section. So it was effectively ignored by the system as all the scores there were similar. This shows how the system is robust to rotations that are not inline with the detected rotation

The parameter extraction was similar to other experiments with spacing and corner trees being acceptably detected. The row count algorithm performed significantly worse on this experiment than the previous experiment detecting a maximum of 8 rows. But we already know from previous experiments that this algorithm is insufficient when dealing with real-world data. There is a lot of variance in the inter-row spacing but that can be attributed to the small differences in rotation in each execution. The nearest neighbour and angle classification may cause the points to change from being just in range to just out of range.

The main point of this experiment was a success. Which is that it could correctly and definitively identify the correct pattern in a dataset with multiple rotations. There was a worry that by rotating the rectangles enough we would find that one of the triangle templates were detected as a false positive due to a large amount of lower correlation matching. But this isn't an issue because a single higher correlation match will have lower correlation matches around it boosting the score significantly and this has far more of an effect than noise does on the dataset. So the algorithm is robust to noise and other rotations.

**41630 Appendix B3**
*Results in Appendix C4*

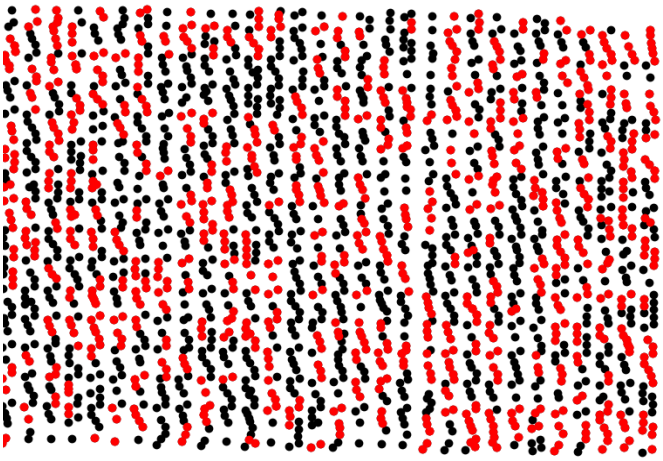


*Image 7: 41630 Row Detection*

This dataset was run because it violates all the assumptions about the data. The row detection minimum length had to be reduced to at least 6 points for this to work (And even this it struggled as can be seen above). But we can see in the image above how many rows were attempted to be detected and how the row detection is vulnerable to noise (Red point clusters are rows). This shows the shortcomings of the row detection algorithm used by the system and the need for a more complex algorithm when dealing with more complex patterns and pattern variations.

The pattern detected was a rectangle. The actual pattern is a variation of the double hedgerow that has 3 plants and then a space (triple hedgerow). Once again we see the template similarity problem at work here, and the main reason the square template wasn't also found was because of the penalty based on length. But with the amount of noise present in the dataset, it seems more like a random assortment of points than an actual pattern. The finding of the rectangle here can be considered a success, or at least not a failure. The double hedgerow pattern is another pattern that could have been found but between the template similarity issue and the extreme amount of noise in the data, unsurprisingly, it was not detected. A failure for this data set would have been the detection of the triangle template.

There is little to say for parameter extraction on this dataset. There is so much noise that it is impossible to determine if the results are correct or not. Except for corner trees (correct), tree count (correct) and roads detected(incorrect). The variation for the other results is too great to determine. The fact that inter-row spacing is greater than intra-row seems correct on a visual inspection of the data, but determining the true space between rows has too much variation.

This experiment was mostly a test to see how the program deals with noise and it went well. The pattern detected was satisfactory or at least not incorrect. So even if there is no clearly discernible pattern the program does not have nonsense output line non-similar patterns (e.g. triangle pattern and double hedgerow) or output every possible pattern.

**35516 Appendix B5**
*Results in Appendix C5*

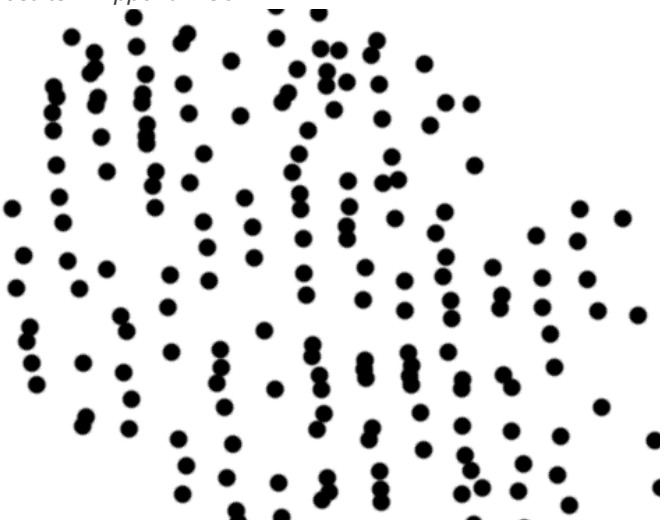


*Image 8: 35516 Noise*

This dataset also has an extreme amount of noise and was used to test the robustness of the program. The pattern detected in every experiment was the triangle pattern. With a secondary pattern of the rectangle pattern being detected twice. Based on a zoomed out visual inspection one could classify the correct planting pattern as the rectangle pattern. But when zoomed in, the extreme level of noise would warrant both rectangle and triangle as the correct answer. This is an important experiment because not only is it the only experiment that does not have a rectangle as the detected pattern. It shows that there is a level of noise that can confuse patterns for template matching. When comparing Image 7 to

appendix B5 it looks like completely different datasets. The problem here with noise is twofold: the drift of points away from true position and missing points confusing the pattern answer. This does show that template matching is robust to noise because not only did we not get a nonsense result. The result was consistent across all five experiments which shows that it did not simply happen by chance and is a credit to the robustness of the system.

Due to the level of noise present in the system, it would be unreasonable to assume any of the parameters are correct. But it would also be unreasonable to attribute the failure to the system as the assumptions that the system is designed for have been compromised by this dataset.

**4.4 Performance and memory**

The performance of the tests was satisfactorily ranging from 30 to 104 seconds is a reasonable execution time. This is within acceptable bounds for a program dealing with 2000 to 5000 data points. This is taking into account that the experiments were run on an 1TB HDD laptop running Ubuntu. With 12 gb DDR3 RAM, Intel i7 CPU, GTX GeForce 1050 GPU.

There are minimal memory requirements for the system to work. although the noise present in the data increases so does the amount of generated templates. Templates such as square and quincunx which only take in a single parameter of length will scale linearly with the increase in possible scales. Templates such as Double Hedge Row and Rectangle take in two length parameters and these will scale exponentially with the detection of possible scales. The number of templates can be calculated as nC2 where n is the length of the scales array. Although each template scales from 200 to 500 bytes (based on spacing parameters) this is not a major issue for the program.

**4.5 Limitations**

The experiment was limited by the data provided. Of the 7 datasets, 2 were unusable (Appendix B6 and B7), 2 had massive violations of assumptions, 1 had a minor assumption violation, 1 lacked too many points to determine a pattern and the last 2 were useful for the program. However, in all 7, 4 of the patterns present were rectangles which made real-world testing of the other patterns impossible (The last one is indeterminate). The datasets that didn't have a rectangle pattern had very curved rows or massive amounts of noise. Going forward it would be useful to test this system on more real-world data to fully learn its capabilities.

**5. Conclusions and Future Work**

From the real-world experiments, we can determine that the template matching algorithm successfully works and the primary aim of this project was successfully completed. This constitutes two findings: firstly the solutions implemented for the template matching issues of scale and rotation successfully identify the correct value(s). Secondly, template matching is a viable solution for detecting planting patterns in a real-world context. This work can be furthered by dealing with multiple rotations, curved rows and height dependent patterns. We have also found areas where the template matching approach may not work. These are for variations of planting patterns that are very similar to each other (e.g. double and triple hedgerow). The issue that applies to the quincunx can appear when templates are too similar so template matching may only be able to offer a broad categorisation but this needs to be investigated further. That said most planting patterns do not have particularly similar variations to them so template matching is an acceptable approach to detecting patterns.

The secondary aim of the project was a partial success. The parameters that were successfully extracted were corner tree coordinates, intra-row spacing and tree count. With inter-row spacing being a partial success as there are datasets where it has a lot of variance to the answer but is sufficiently close to the true answer. The parameters for row count, average trees per row and road count are considered failures by the criteria of the secondary aim. This is because they either did not fully detect the correct values or did not detect the value(s) at all. This is acceptable however as all of these parameters can be solved accurately by implementing more complex row detection algorithms.

The project as a whole was a success as it fulfilled its primary aim and even managed partial success on the secondary aim. In the future, going forward with this method it can be extended to deal with more complex datasets that contain curved rows or multiple planting patterns. Noise reduction techniques can be implemented such as realigned points onto a row to increase the confidence of the results for template matching.

**6. Acknowledgements**

We would like to thank our project supervisor, Associate Professor Patrick Marais, for the guidance and aid he gave during the project. We would also like to thank Aerobotics for providing us with the data used in the project.

**8. References**

[1] Zhang, C., Valente, J., Kooistra, L., Guo, L., & Wang, W. (2021). Orchard management with small unmanned aerial vehicles: a survey of sensing and analysis approaches. *Precision Agriculture*, 1-46.

[2] Zarea, M. J., Ghalavand, A., & Daneshian, J. (2005). Effect of planting patterns of sunflower on yield and extinction coefficient. *Agronomy for sustainable development*, *25*(4), 513-518.

[3] Lyon, D. J., Pavlista, A. D., Hergert, G. W., Klein, R. N., Shapiro, C. A., Knezevic, S., ... & Aiken, R. M. (2009). Skip‐row planting patterns stabilize corn grain yields in the central Great Plains. *Crop Management*, *8*(1), 1-8.\

[4] GOSHTASBY A. 1985. Template matching in rotated images. IEEE Transactions on Pattern Analysis and Machine Intelligence, (3), 338-344.

[5] Olofsson, K., Wallerman, J., Holmgren, J., & Olsson, H. (2006). Tree species discrimination using Z/I DMC imagery and template matching of single trees. *Scandinavian Journal of Forest Research*, *21*(S7), 106-110.

[6] Bhatia, N. (2010). Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085*.

[7] ZAINUDDIN, N. E., AND DALIMAN, S. 2020. Analysis of Rubber Tree Recognition Based on Drone Images. In IOP Conference Series: Earth and Environmental Science (Vol. 549, No. 1, p. 012012). IOP Publishing.

[8] KALANTAR, B., MANSOR, S. B., SHAFRI, H. Z. M., AND HALIN, A. A. 2016. Integration of template matching and object-based image analysis for semiautomatic oil palm tree counting in UAV images. In Proceedings of the 37th Asian Conference on Remote Sensing, Colombo, Sri Lanka pp. 17-21.

[9] GRUEN, A. 2012. Development and status of image matching in photogrammetry. The Photogrammetric Record, 27(137), 36-57.

[10] DI STEFANO, L., AND MATTOCCIA, S. 2003, September. A sufficient condition based on the Cauchy-Schwarz inequality for efficient template matching. In Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429) Vol. 1, pp. I-269. IEEE. Template Matching Cape Town 2021 June

[11] SWAROOP, P., AND SHARMA, N. 2016. An overview of various template matching methodologies in image processing. International Journal of Computer Applications, 153(10), 8-14.

[12] OUYANG, W., TOMBARI, F., MATTOCCIA, S., DI STEFANO, L., AND CHAM, W. K. 2011. Performance evaluation of full search equivalent pattern matching algorithms. IEEE transactions on pattern analysis and machine intelligence, 34(1), 127-143.

[13] DEKEL, T., ORON, S., RUBINSTEIN, M., AVIDAN, S., AND13 FREEMAN, W. T. 2015. Best-buddies similarity for robust template matching. In Proceedings of the IEEE conference on computer vision and pattern recognition pp. 2021-2029.

[14] BRIECHLE, K., & HANEBECK, U. D. 2001. Template matching using fast normalized cross correlation. In Optical Pattern Recognition XII (Vol. 4387, pp. 95-102). International Society for Optics and Photonics.

[16] Basulto-Lantsova, A., Padilla-Medina, J. A., Perez-Pinal, F. J., & Barranco-Gutierrez, A. I. (2020, January). Performance comparative of OpenCV Template Matching method on Jetson TX2 and Jetson Nano developer kits. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0812-0816). IEEE.

[17] Thakar, K., Kapadia, D., Natali, F., & Sarvaiya, J. (2017). Implementation and analysis of template matching for image registration on DevKit-8500D. *Optik*, *130*, 935-944.

[18] HISHAM, M. B., YAAKOB, S. N., RAOF, R. A., AND NAZREN, A. A. 2015. Template matching using sum of squared difference and normalized cross correlation. In 2015 IEEE student conference on research and development (SCOReD) pp. 100-104. IEEE.

[19] KERTÉSZ, G., SZÉNÁSI, S., AND VÁMOSSY, Z. 2015. Performance measurement of a general multi-scale template matching method. In 2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES) pp. 153-157. IEEE.

[20] YANG, Y., CHEN, Z., LI, X., GUAN, W., ZHONG, D., AND XU, M. 2020. Robust template matching with large angle localization. Neurocomputing, 398, 495-504.

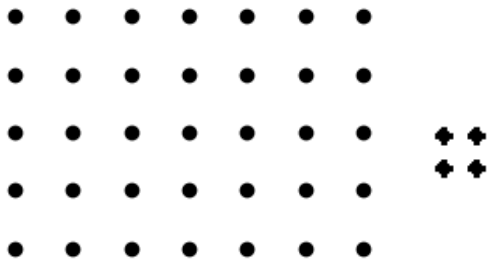
[21] MATTOCCIA, S., TOMBARI, F., AND DI STEFANO, L. 2011. Efficient template matching for multi-channel images. Pattern Recognition Letters, 32(5), 694-700.

[22] TALMI, I., MECHREZ, R., AND ZELNIK-MANOR, L. 2017. Template matching with deformable diversity similarity. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition pp. 175-183.

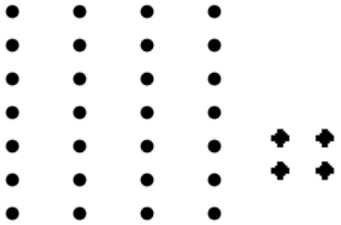
[23] Jurie, F., & Dhome, M. (2002, September). Real Time Robust Template Matching. In *BMVC* (Vol. 2002, pp. 123-132).

Appendix A Ideal Pattern examples

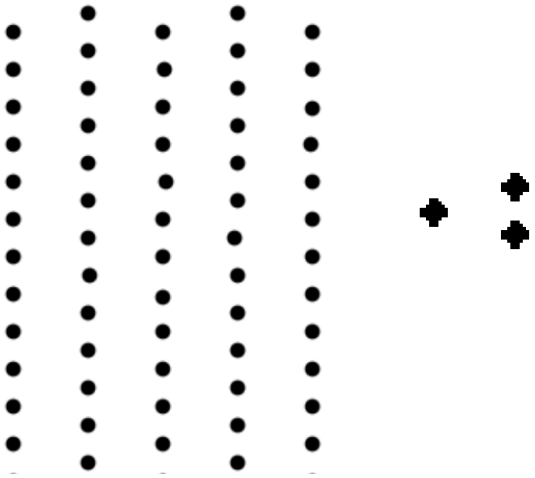
A1 Square Pattern



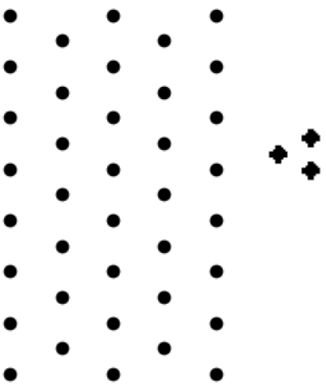
A2 Rectangle Pattern



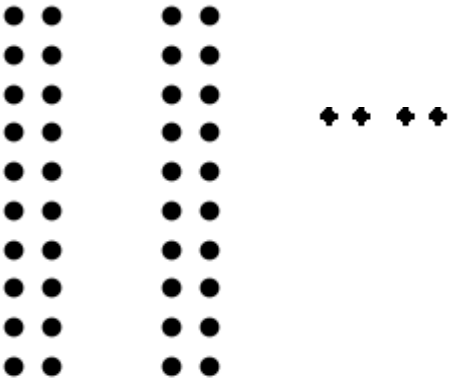
A3 Triangle Pattern



A4 Hexagonal (Equilateral Triangle) Pattern

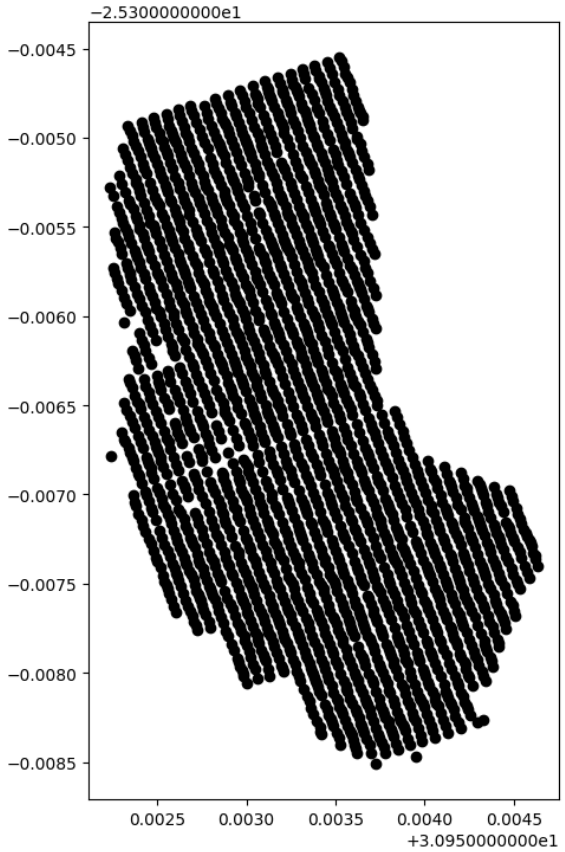


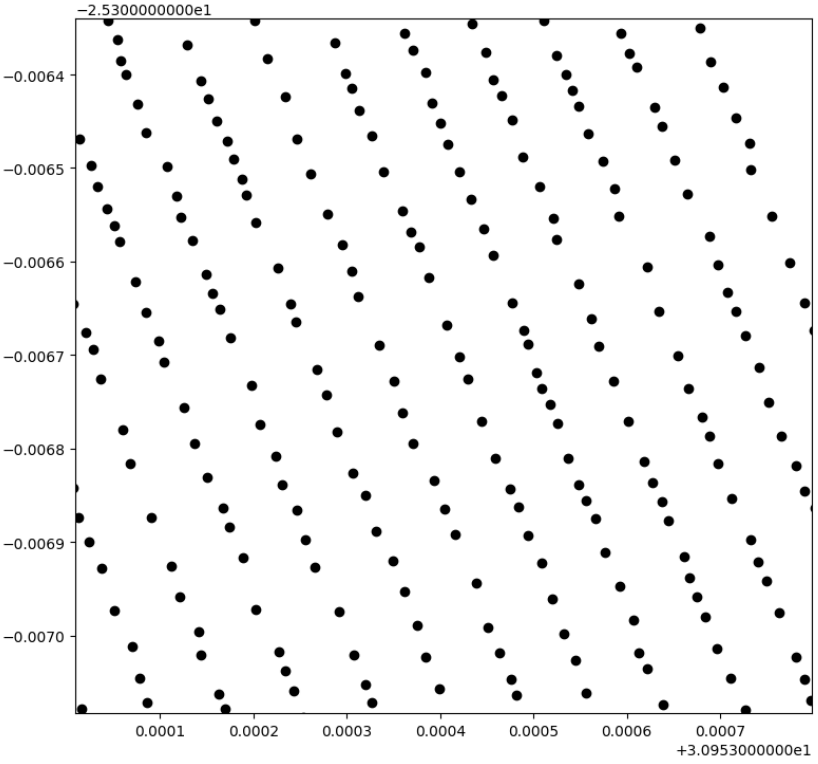
A6 Double Hedge Row Pattern



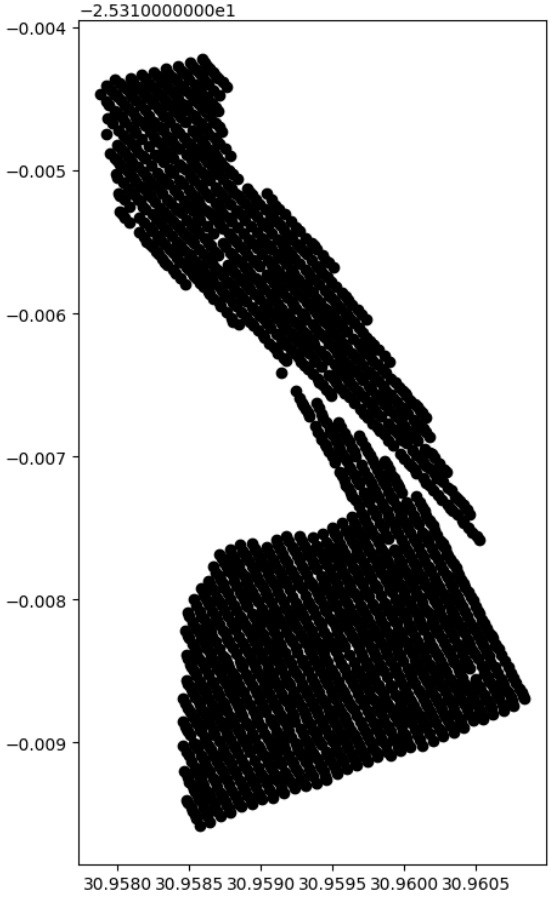
## Appendix B GeoJSON DataSet Visualisations

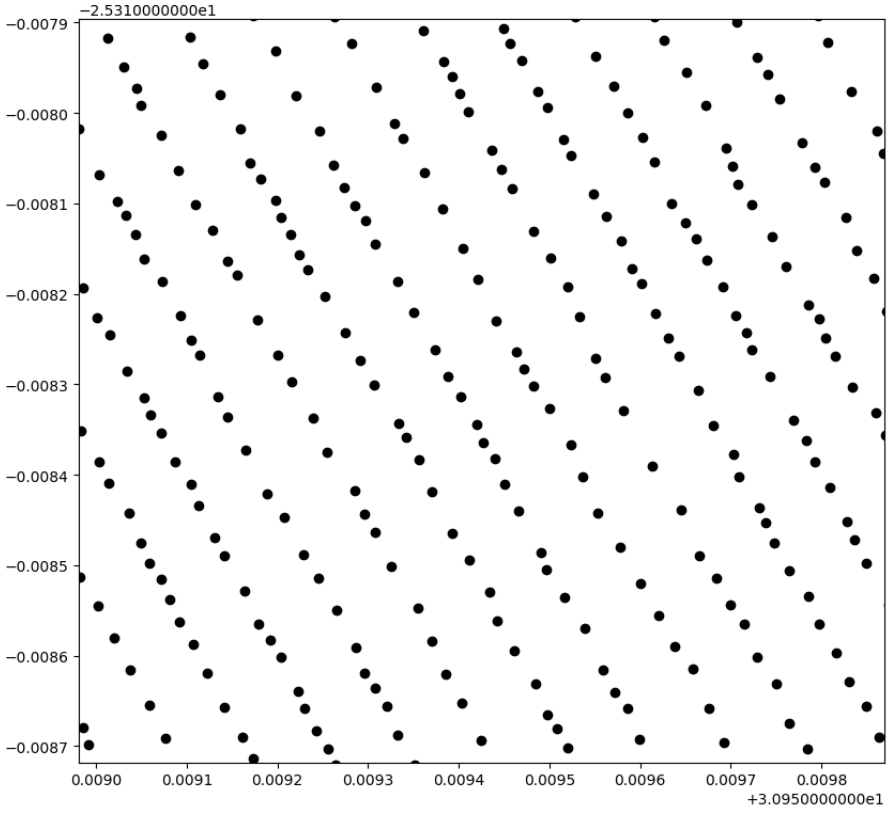
**B1 - 36507**



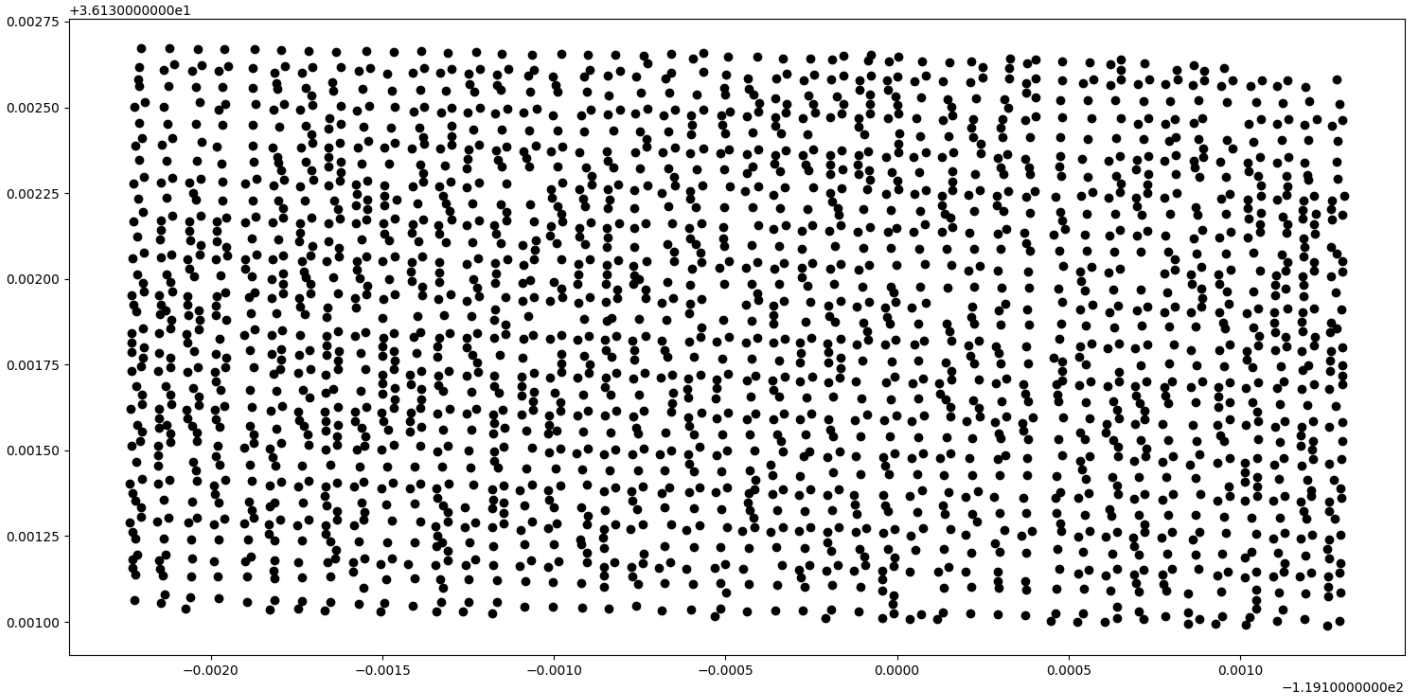



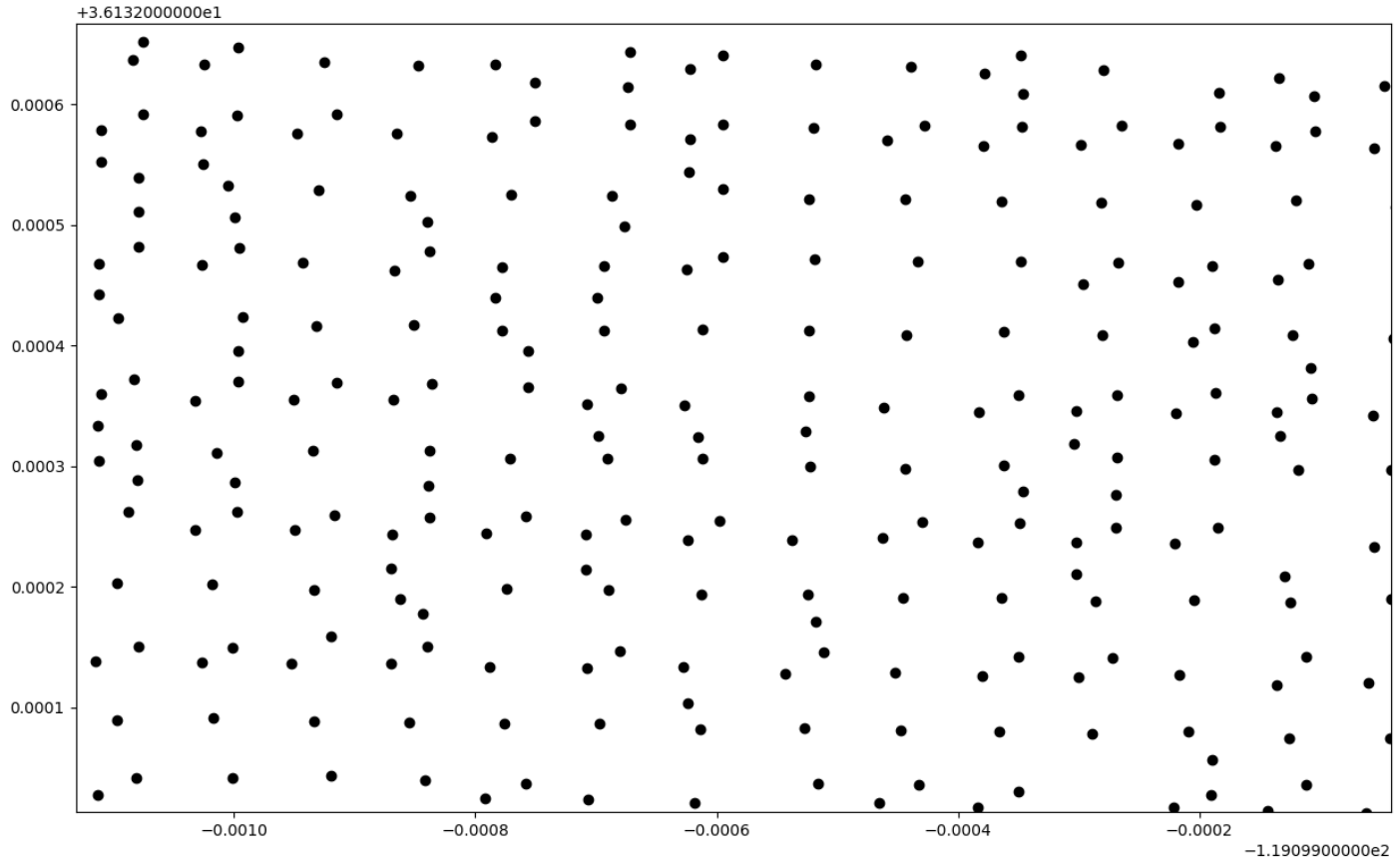
**B2 - 36513**


−2.5310000000e1
−0.004
−0.005
−0.006
−0.007
−0.008
−0.009
30.9580 30.9585 30.9590 30.9595 30.9600 30.9605

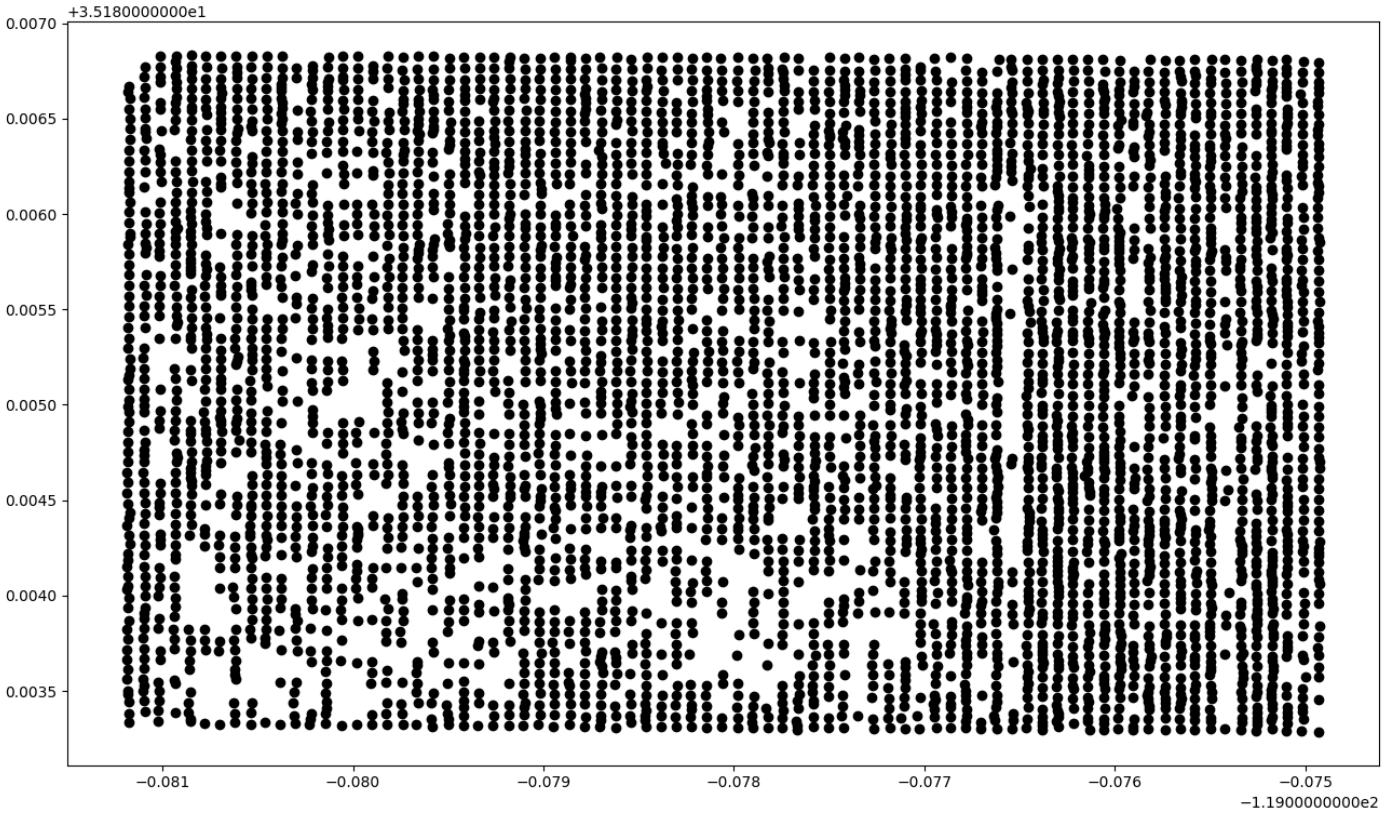

−2.5310000000e1
−0.0079
−0.0080
−0.0081
−0.0082
−0.0083
−0.0084
−0.0085
−0.0086
−0.0087
0.0090 0.0091 0.0092 0.0093 0.0094 0.0095 0.0096 0.0097 0.0098
+3.0950000000e1

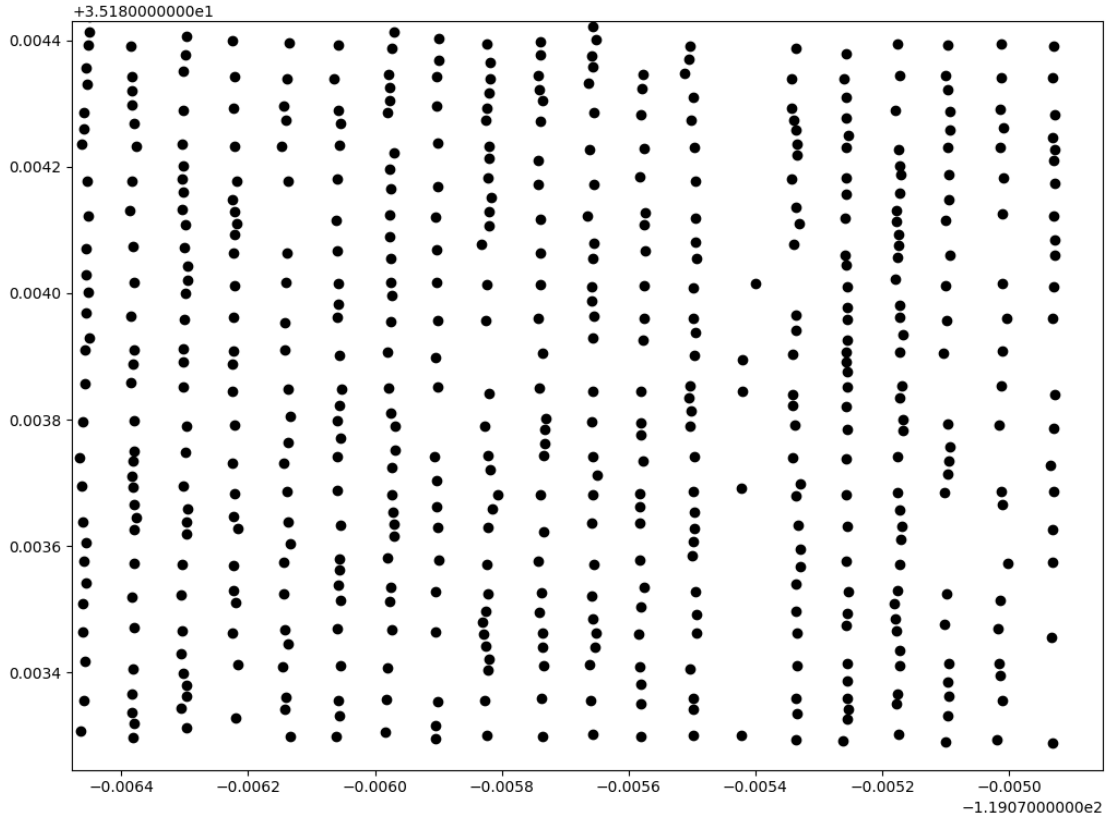
**B3 - 41630**


+3.6130000000e1
0.00275
0.00250
0.00225
0.00200
0.00175
0.00150
0.00125
0.00100
−0.0020
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
−1.1910000000e2

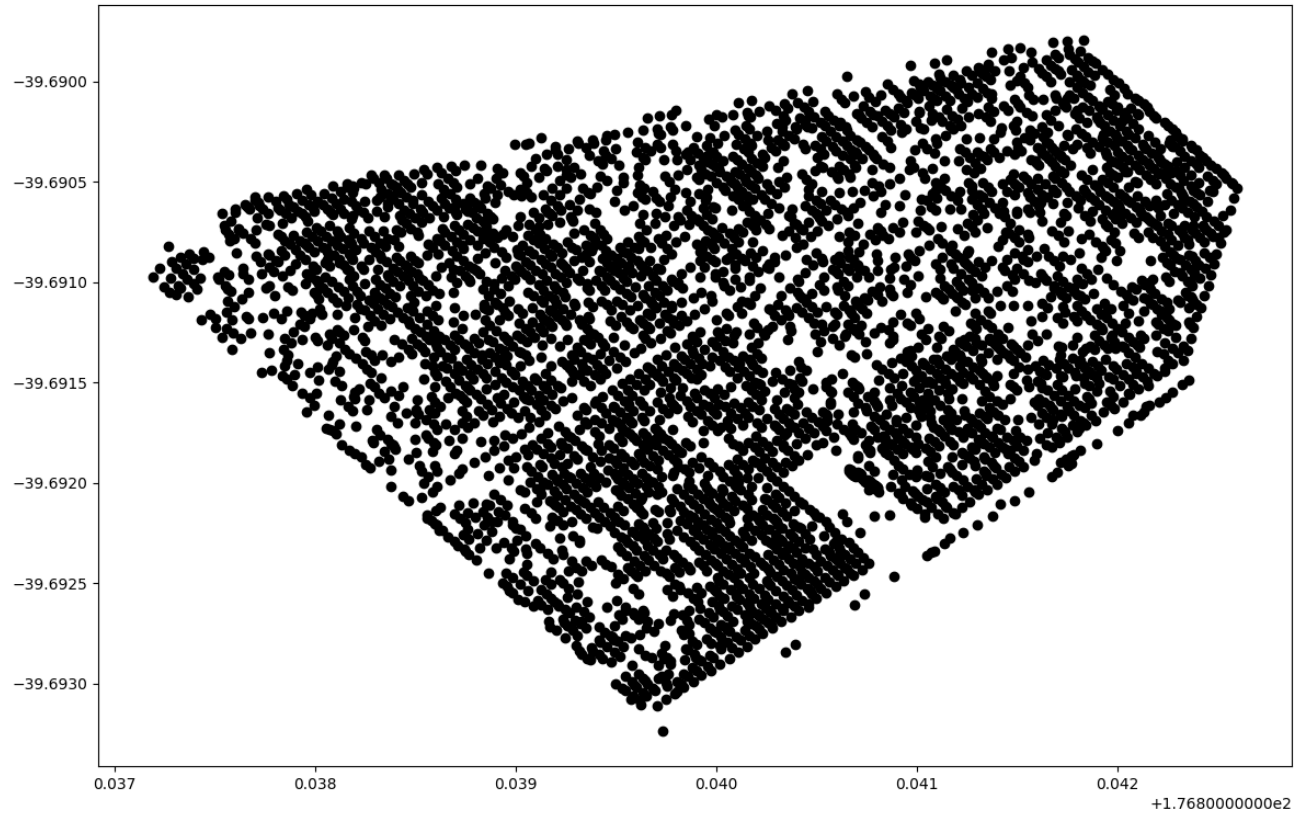

+3.6132000000e1
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
−0.0010
−0.0008
−0.0006
−0.0004
−0.0002
−1.1909900000e2

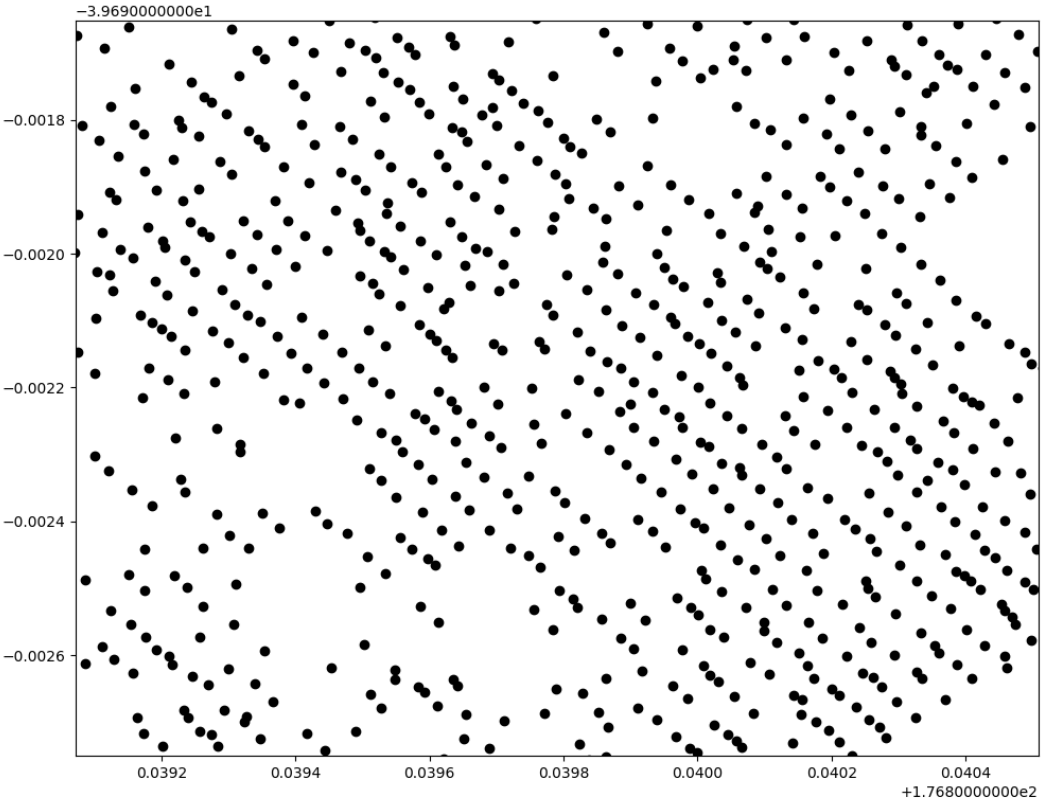
**B4 - 43581**


+3.5180000000e1
0.0070
0.0065
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
−0.081
−0.080
−0.079
−0.078
−0.077
−0.076
−0.075
−1.1900000000e2

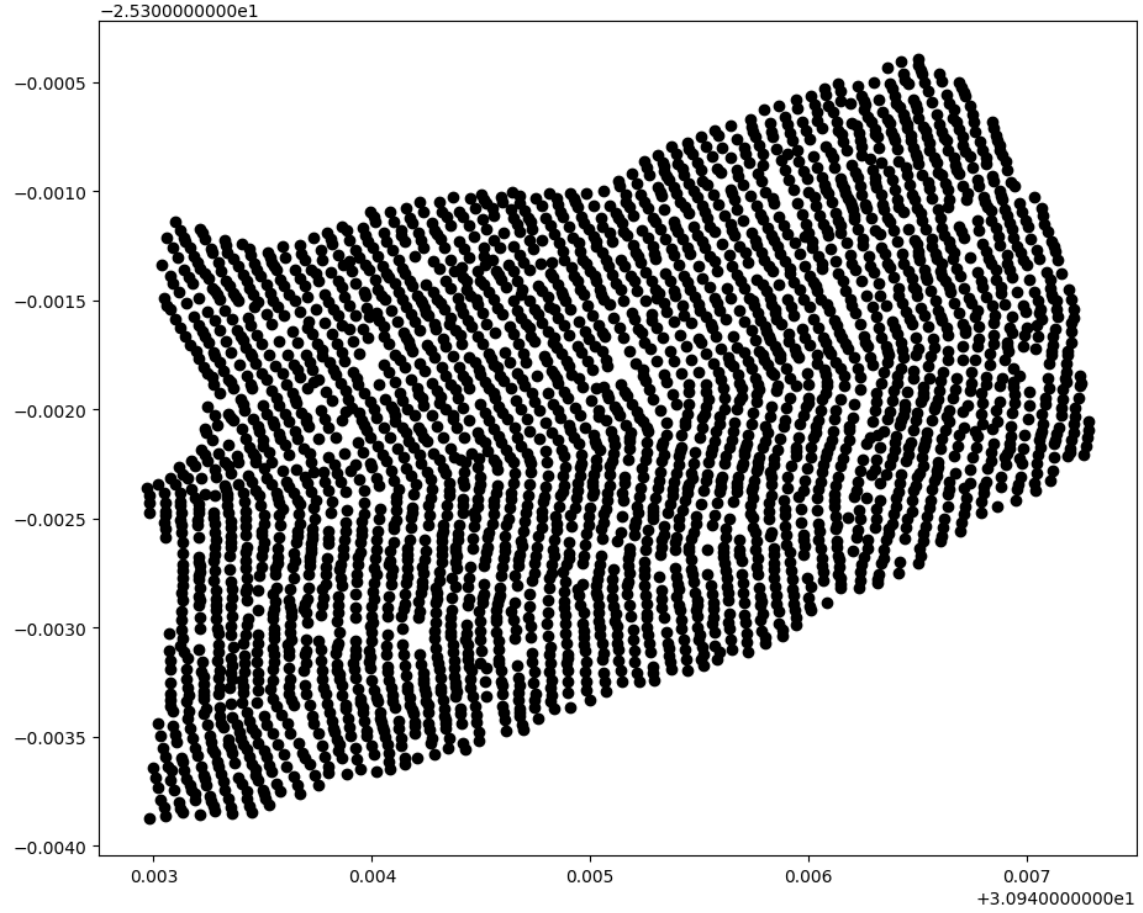

+3.5180000000e1
0.0044
0.0042
0.0040
0.0038
0.0036
0.0034
−0.0064
−0.0062
−0.0060
−0.0058
−0.0056
−0.0054
−0.0052
−0.0050
−1.1907000000e2

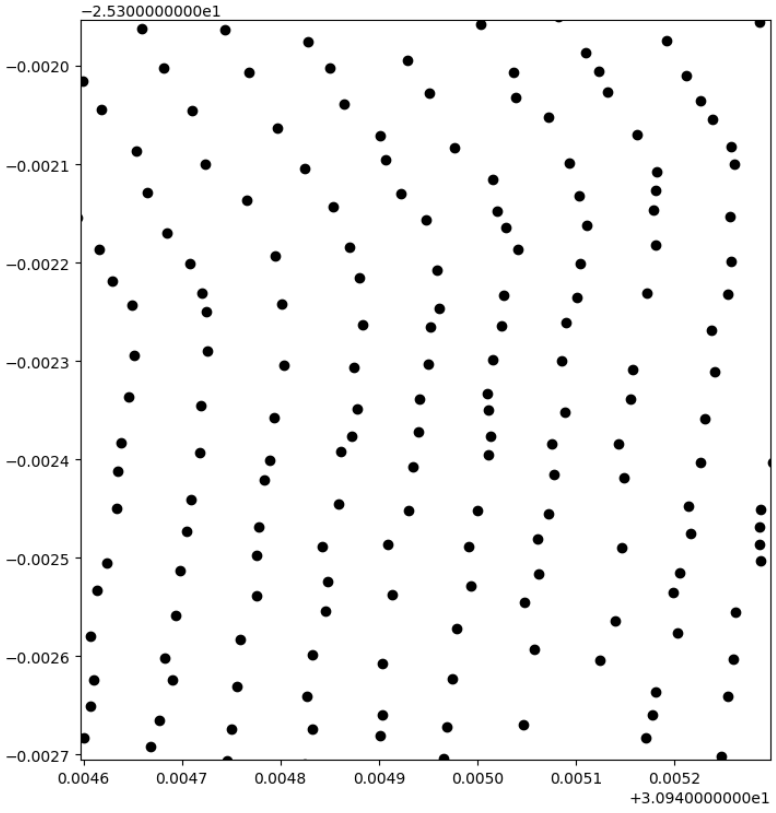
## B5 - 35516


−39.6900
−39.6905
−39.6910
−39.6915
−39.6920
−39.6925
−39.6930
0.037
0.038
0.039
0.040
0.041
0.042
+1.7680000000e2

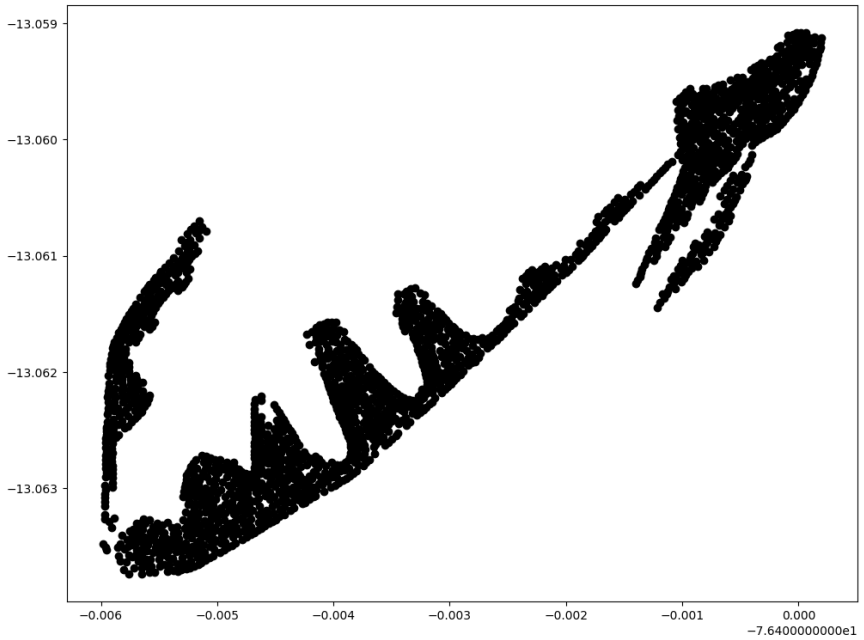

−3.9690000000e1
−0.0018
−0.0020
−0.0022
−0.0024
−0.0026
0.0392
0.0394
0.0396
0.0398
0.0400
0.0402
0.0404
+1.7680000000e2

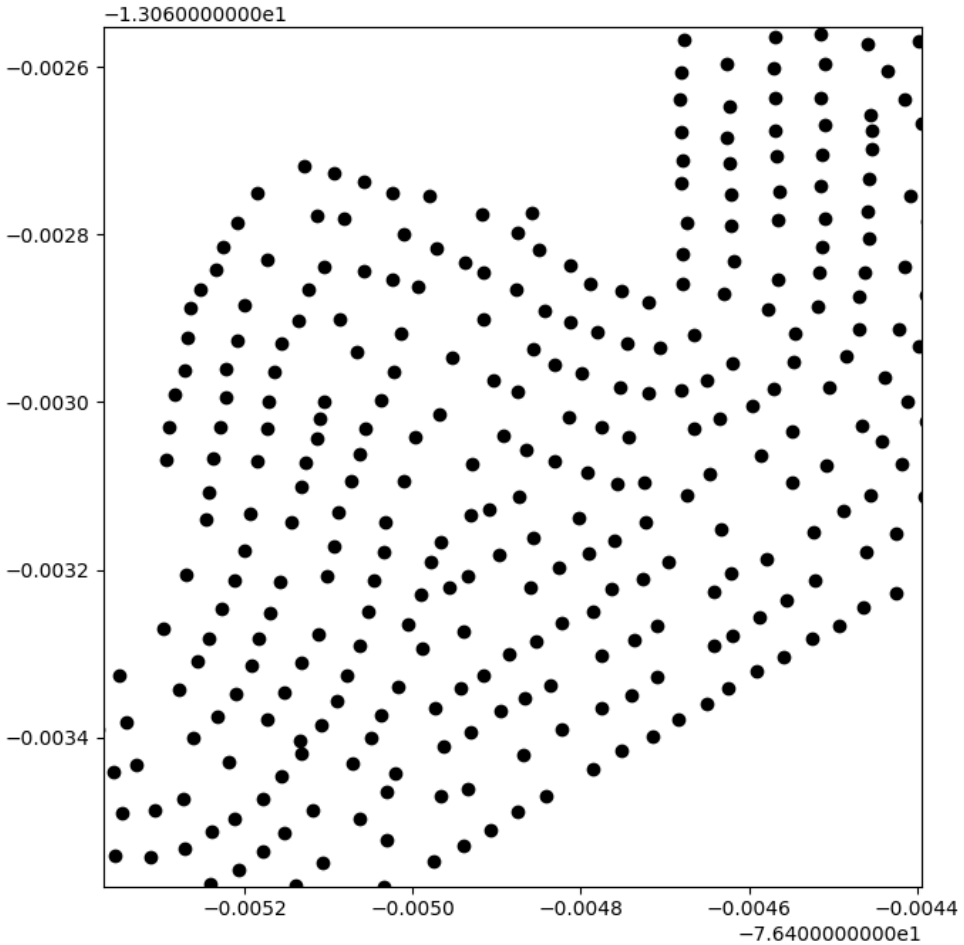
**B6 - 36502** Unusable Dataset due to multiple rotations in a row



**B7 - 32377** Unusable Dataset due to curved rows, inconsistent clusters, no discernable pattern



**Appendix C - Idealised Experiment output**

**C1 Square**
The intra-row spacing mean value is: 9.8384936060954 m
The inter-row spacing mean value is: 10.353669708955517 m
The Tree count detected is: 2500
The number of rows detected is: 50
The mean value per row is: 14
There were 0 road(s) or ditches detected
A corner coordinate is: -0.8 , 390.0
A corner coordinate is: 160.0 , -1.0
A corner coordinate is: 490.8 , 90.0
A corner coordinate is: 190.0 , 491.0
The detected pattern(s) are Square

**C2 Rectangle**
The intra-row spacing mean value is: 9.917860090149325 m
The inter-row spacing mean value is: 21.36378897810072 m
The Tree count detected is: 1250
The number of rows detected is: 5
The mean value per row is: 42
There were 0 road(s) or ditches detected
A corner coordinate is: -0.9 , 230.0
A corner coordinate is: 140.0 , -0.4
A corner coordinate is: 480.0 , 0.0
A corner coordinate is: 0.0 , 490.0
The detected pattern(s) are Rectangle

**C3 Triangle**
The intra-row spacing mean value is: 9.992006394884093 m
The inter-row spacing mean value is: 22.434914959741658 m
The Tree count detected is: 1251
The number of rows detected is: 25
The mean value per row is: 50
There were 0 road(s) or ditches detected
A corner coordinate is: 0.0 , 0.0
A corner coordinate is: 5.0 , 0.0
A corner coordinate is: 495.0 , 0.0
A corner coordinate is: 5.0 , 480.0
The detected pattern(s) are Triangle

**C4 Hexagonal**
The intra-row spacing mean value is: 9.996001599360255 m
The inter-row spacing mean value is: 10.027756377319946 m
The Tree count detected is: 2501
The number of rows detected is: 1
The mean value per row is: 50
There were 0 road(s) or ditches detected
A corner coordinate is: 0.0 , 0.0
A corner coordinate is: 5.0 , 0.0
A corner coordinate is: 495.0 , 0.0
A corner coordinate is: 0.0 , 490.0
The detected pattern(s) are Hexagon

**C5 Double HedgeRow**
The intra-row spacing mean value is: 4.0 m
The inter-row spacing mean value is: 10.22470924892222 m
The Tree count detected is: 4200
The number of rows detected is: 50
The mean value per row is: 84
There were 0 road(s) or ditches detected
A corner coordinate is: 0.0 , 0.0
A corner coordinate is: 4.0 , 0.0
A corner coordinate is: 496.0 , 0.0
A corner coordinate is: 0.0 , 490.0
The detected pattern(s) are Double Hedge Row, Rectangle

**Appendix D - Real World Experiment output**

**D1 43581**
**Scoring**
*1 output because it was consistent across all executions*

| | | |
|---|---|---|
| Square | 43343.75 | [0, 0, 0, 28, 176, 923, 3093] |
| Rectangle | 345000.0 | [0, 0, 20, 250, 976, 2408, 4983] |
| Isos | 25937.5 | [0, 0, 0, 0, 38, 339, 1456] |
| Equi | 73000.0 | [0, 0, 4, 27, 165, 698, 2362] |
| Dbl | 11312.5 | [0, 0, 0, 0, 12, 157, 914] |

**Execution 1**
The intra-row spacing mean value is: 4.695934284167114 m
The inter-row spacing mean value is: 15.053353562672338 m
The Tree count detected is: 4992
The number of rows detected is: 3
The mean value per row is: 37
There were 0 road(s) or ditches detected
A corner coordinate is: -119.08118786864887 , 35.18454024924922
A corner coordinate is: -119.07493100160913 , 35.18328813206948
A corner coordinate is: -119.07492656836743 , 35.18466585878412
A corner coordinate is: -119.08077237486928 , 35.18683034037097
The detected pattern(s) are Rectangle
--- 98.02153468132019 seconds ---

**Execution 2 differences**

The inter-row spacing mean value is: 11.927847846094117 m
The Tree count detected is: 4992
The number of rows detected is: 5
The mean value per row is: 43
--- 105.64432144165039 seconds ---

**Execution 3 differences**
The inter-row spacing mean value is: 11.68085422434415 m
The number of rows detected is: 6
The mean value per row is: 55
--- 102.21804976463318 seconds ---
**Execution 4 differences**

The inter-row spacing mean value is: 11.918935057431028 m
The number of rows detected is: 7
The mean value per row is: 43
-- 105.64496779441833 seconds ---

**Execution 5 differences**

The inter-row spacing mean value is: 13.000070519105591 m
The number of rows detected is: 4
The mean value per row is: 57
There were 0 road(s) or ditches detected
--- 108.83793687820435 seconds ---

**D2 36507**

**Constant values**
*1 output because it was consistent across all executions*

Square 61062.5 [0, 0, 0, 9, 153, 1612, 5958]
Rectangle 225187.5 [0, 0, 4, 51, 488, 2391, 6413]
Isos 17937.5 [0, 0, 0, 7, 16, 227, 761]
Equi 36312.5 [0, 0, 0, 1, 23, 531, 2324]
Dbl 3687.5 [0, 0, 0, 0, 8, 43, 170]

The intra-row spacing mean value is: 3.103828590536644 m
The Tree count detected is: 2412
There were 0 road(s) or ditches detected
A corner coordinate is: 30.952235789689155 , -25.305281818730094
A corner coordinate is: 30.953726134711317 , -25.30851159348046
A corner coordinate is: 30.9546363997195 , -25.307401171514496
A corner coordinate is: 30.953531669671214 , -25.304570885590223

**Execution 1**

The inter-row spacing mean value is: 8.583340234595086 m
The number of rows detected is: 16
The mean value per row is: 120
The detected pattern(s) are Rectangle
--- 37.601715326309204 seconds ---

**Execution 2 differences**

The inter-row spacing mean value is: 8.526844272969281 m
The number of rows detected is: 15
The mean value per row is: 88
There were 0 road(s) or ditches detected
--- 40.25502109527588 seconds ---

**Execution 3 differences**

The inter-row spacing mean value is: 8.715946180646114 m
The number of rows detected is: 17
The mean value per row is: 95
--- 39.91319751739502 seconds ---
**Execution 4 differences**

The inter-row spacing mean value is: 8.80535590014104 m
The Tree count detected is: 2412
The number of rows detected is: 13
The mean value per row is: 86
--- 39.86967754364014 seconds ---

**Execution 5 differences**

The inter-row spacing mean value is: 8.177426382170303 m
The number of rows detected is: 13
The mean value per row is: 84
--- 40.69258165359497 seconds ---

**D3 36513**

**Constant Values**
Square 107437.5 [0, 0, 0, 0, 244, 2950, 8773]
Rectangle 317625.0 [0, 0, 1, 59, 563, 3712, 9508]
Isos 35625.0 [0, 0, 0, 0, 4, 562, 2936]
Equi 95250.0 [0, 0, 0, 0, 38, 1448, 3720]
Dbl 187.5 [0, 0, 0, 0, 0, 3, 21]

The detected pattern(s) are Rectangle

The intra-row spacing mean value is: 3.276327607718036 m
There were 0 road(s) or ditches detected
The Tree count detected is: 2368
A corner coordinate is: 30.957879211217218 , -25.31446977119233
A corner coordinate is: 30.958569924333194 , -25.31958096394645
A corner coordinate is: 30.960836256684527 , -25.318694972569546
A corner coordinate is: 30.95850325954219 , -25.31424639226457

**Execution 1**
The inter-row spacing mean value is: 9.753264452137339 m
The number of rows detected is: 8
The mean value per row is: 87
--- 29.49584150314331 seconds ---

**Execution 2 differences**
The inter-row spacing mean value is: 13.377890741480659 m
The number of rows detected is: 3
The mean value per row is: 48
--- 32.0698459148407 seconds ---

**Execution 3 differences**
The inter-row spacing mean value is: 13.99085185369315 m
The number of rows detected is: 3
The mean value per row is: 50
--- 30.19421100616455 seconds ---

**Execution 4 differences**
The inter-row spacing mean value is: 9.425204191636874 m
The number of rows detected is: 7
The mean value per row is: 69
--- 29.630834341049194 seconds ---

**Execution 5 differences**
The inter-row spacing mean value is: 15.481203118109452 m
The number of rows detected is: 2
The mean value per row is: 44
--- 31.43457865715027 seconds ---

**D4 41630 0(.3)**
**Constant Values**
The intra-row spacing mean value is: 2.9864772835455935 m
The Tree count detected is: 2891
There were 0 road(s) or ditches detected
A corner coordinate is: -119.10223745691691 , 36.131291184961384
A corner coordinate is: -119.09874566338804 , 36.130988250867745
A corner coordinate is: -119.09869690394792 , 36.13224244283178
A corner coordinate is: -119.1021218875335 , 36.132671788946986

**Execution 1**
Square 2218.75 [0, 0, 0, 4, 13, 29, 657]
Rectangle 14687.5 [0, 0, 0, 8, 36, 131, 496]
Isos 4375.0 [0, 0, 0, 0, 13, 44, 158]
Equi 10625.0 [0, 0, 2, 7, 23, 80, 275]
Dbl 625.0 [0, 0, 0, 0, 0, 10, 72]
The inter-row spacing mean value is: 12.446656108340994
The number of rows detected is: 2
The mean value per row is: 8
The detected pattern(s) are Rectangle
--- 68.15329462875318 seconds ---

**Execution 2**
Square 7343.75 [0, 0, 0, 4, 26, 167, 842]
Rectangle 17312.5 [0, 0, 0, 0, 17, 243, 1516]
Isos 23000.0 [0, 0, 0, 4, 43, 266, 1010]
Equi 26812.5 [0, 0, 0, 0, 70, 289, 920]
Dbl 2937.5 [0, 0, 0, 0, 0, 47, 279]
The inter-row spacing mean value is: 10.925241412127416 m
The number of rows detected is: 3
The mean value per row is: 11
The detected pattern(s) are Triangle, Hexagonal
--- 66.47577667236328 seconds ---

**Execution 3 differences**
Square 10218.75 [0, 0, 0, 0, 26, 275, 1373]
Rectangle 51125.0 [0, 0, 0, 12, 118, 534, 1733]
Isos 20500.0 [0, 0, 1, 4, 42, 220, 906]
Equi 31625.0 [0, 0, 0, 4, 69, 352, 1190]
Dbl 4000.0 [0, 0, 0, 0, 0, 64, 336]
The inter-row spacing mean value is: 17.62015673666584 m
The number of rows detected is: 2
The mean value per row is: 9
The detected pattern(s) are Rectangle
--- 78.1531894324424 seconds ---

**Execution 4 differences**
Square 14718.75 [0, 0, 0, 2, 43, 377, 1573]
Rectangle 64500.0 [0, 0, 0, 23, 161, 618, 1817]
Isos 22062.5 [0, 0, 0, 6, 49, 231, 799]
Equi 42000.0 [0, 0, 0, 10, 112, 408, 1293]
Dbl 3750.0 [0, 0, 0, 0, 1, 58, 333]
The inter-row spacing mean value is: 10.486713256060932 m
The number of rows detected is: 4
The mean value per row is: 10
The detected pattern(s) are Rectangle
--- 72.69970917701721 seconds ---

**Execution 5 differences**
Square 2156.25 [0, 0, 0, 1, 9, 47, 206]
Rectangle 15875.0 [0, 0, 0, 7, 28, 170, 604]
Isos 4062.5 [0, 0, 0, 0, 9, 47, 212]
Equi 10562.5 [0, 0, 1, 8, 27, 75, 293]
Dbl 2312.5 [0, 0, 0, 0, 0, 37, 155]
The inter-row spacing mean value is: 14.996362634626196 m
The number of rows detected is: 2
The mean value per row is: 9
The detected pattern(s) are Rectangle
--- 77.08587121963501 seconds ---

**D5 35516**
**Constant Values**
The intra-row spacing mean value is: 3.2171418552456417 m
The Tree count detected is: 4280
There were 0 road(s) or ditches detected
A corner coordinate is: 176.83719197967946 , -39.69097308221957
A corner coordinate is: 176.8397303544299 , -39.693237953853895
A corner coordinate is: 176.8425974886896 , -39.690534484209664
A corner coordinate is: 176.841750678421 , -39.68979909034605

**Execution 1**
Square 49500.0 [0, 0, 0, 0, 25, 1534, 2914]
Rectangle 99000.0 [0, 0, 0, 0, 25, 1534, 2914]
Isos 177500.0 [0, 0, 0, 10, 476, 1848, 3704]
Equi 78687.5 [0, 0, 0, 4, 36, 1171, 2988]
Dbl 111125.0 [0, 0, 1, 33, 175, 1288, 5971]
The inter-row spacing mean value is: 13.197782292837157 m
The number of rows detected is: 3
The mean value per row is: 20
The detected pattern(s) are Triangle
--- 70.46583771705627 seconds ---

**Execution 2**
Square 52750.0 [0, 0, 0, 0, 32, 1624, 3053]
Rectangle 93312.5 [0, 0, 0, 0, 0, 1493, 2761]
Isos 152687.5 [0, 0, 0, 4, 370, 1687, 3560]
Equi 79312.5 [0, 0, 0, 3, 28, 1201, 3106]
Dbl 46500.0 [0, 0, 0, 3, 45, 642, 4624]
The inter-row spacing mean value is: 15.943865880143838 m
The number of rows detected is: 2
The mean value per row is: 34
The detected pattern(s) are Triangle
--- 60.08768343925476 seconds ---

**Execution 3**
Square 51906.25 [0, 0, 0, 0, 15, 1631, 2621]
Rectangle 103812.5 [0, 0, 0, 0, 15, 1631, 2621]
Isos 136250.0 [0, 0, 0, 0, 439, 1302, 2560]
Equi 73375.0 [0, 0, 0, 0, 0, 1174, 2881]
Dbl 49875.0 [0, 0, 0, 0, 72, 654, 4177]
The inter-row spacing mean value is: 6.801843949304538 m
The number of rows detected is: 4
The mean value per row is: 18
The detected pattern(s) are Rectangle, Triangle
--- 70.2305736541748 seconds ---

**Execution 4**
Square 52750.0 [0, 0, 0, 0, 32, 1624, 3053]
Rectangle 93312.5 [0, 0, 0, 0, 0, 1493, 2761]
Isos 152687.5 [0, 0, 0, 4, 370, 1687, 3560]
Equi 79312.5 [0, 0, 0, 3, 28, 1201, 3106]
Dbl 46500.0 [0, 0, 0, 3, 45, 642, 4624]
The inter-row spacing mean value is: 14.392794728136554 m
The number of rows detected is: 2
The mean value per row is: 23
The detected pattern(s) are Triangle
--- 60.22697305679321 seconds ---

**Execution 5**
Square 51906.25 [0, 0, 0, 0, 15, 1631, 2621]
Rectangle 103812.5 [0, 0, 0, 0, 15, 1631, 2621]
Isos 136250.0 [0, 0, 0, 0, 439, 1302, 2560]
Equi 73375.0 [0, 0, 0, 0, 0, 1174, 2881]
Dbl 49875.0 [0, 0, 0, 0, 72, 654, 4177]
The inter-row spacing mean value is: 7.348078340226431 m
The number of rows detected is: 12
The mean value per row is: 23
The detected pattern(s) are Rectangle, Triangle
--- 71.3396909236908 seconds ---