Detection of Road Gully Drains

Claudiu Andrei Tarlungianu

4th Year Project Report
Computer Science and Management Science
School of Informatics
University of Edinburgh
2015

Abstract

This paper presents an approach to detecting rectangle-shaped road gullies from aerial images and investigates the performance of different algorithms and constraints in terms of the total number of drains (we will use the terms manhole, gully and drain interchangeably in this paper) correctly identified. The knowledge available to carry out the detection is an image containing multiple gullies and rough estimates of the their positions and of the road edges. For each drain, the problem consists of three steps: finding candidate lines describing the edge of the road close to the drain (all drains are along the edge of the road, opposite the footpath), finding candidate positions for the location of the drain and using the knowledge gathered from the previous two steps to select the most likely candidate for the drain position. For the first step of detecting the edge, the RANSAC [1] (Random Sample Consensus) algorithm was used, with maxima points based on luminance as inputs. For gully detection, the MSER [2] (Maximally Stable Extremal Regions) algorithm was used, alongside constraints such as size, shape and colour. The analysis is carried out using a 10,000x10,000 pixel satellite image, containing a total of 101 drains and achieves an average of 80.39% detection ($\pm$1.9384 std. dev.) with 12.3% false positives ($\pm$0.221 std. dev.) after 20 runs if detection of very shadowy regions is attempted. If however we choose to minimize false-positives, after determining that our drain is covered by a shadow and choosing not to perform detection, the true-positive percentage would decrease by roughly 6%, and so would the number of false positives.

Acknowledgements

First of all, I would like to thank my supervisor Robert Fisher for his support, guidance and advice throughout this project, especially for his ability to keep me on the right track. I would also like to express my gratitude to Robert Inder from Interactive Information Ltd. for providing the satellite image used for detection, the StackOverflow and ImageJ communities, my friend Marius for proofreading this paper and last but not least to my family for their continuous love and support.

Table of Contents

Contents

Chapter 1

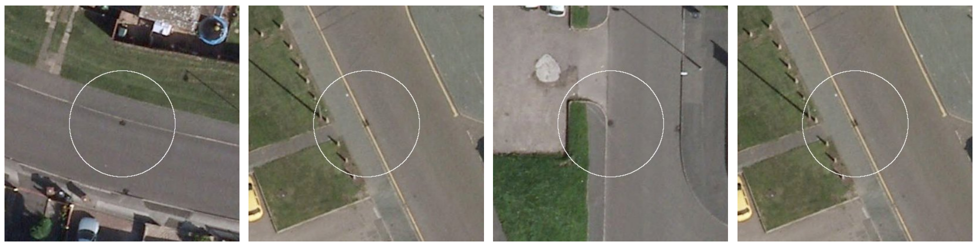
Introduction

1.1 Motivation and Background

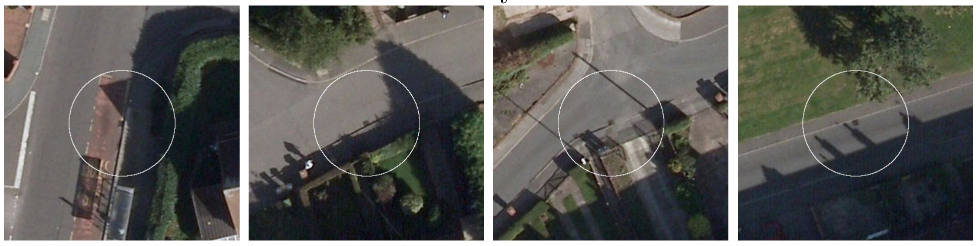
Manhole covers require general maintenance at regular intervals, knowledge of their exact position being necessary for the crew to carry out this maintenance. Nonetheless, the data available might be inaccurate or incomplete. One way to ensure that the available positions are accurate would be to use an automated approach based on image analysis. This paper describes an efficient work-flow for detection of the drains, based on their covers.

The drains available have been initially classified according to how difficult their detection is, depending on lighting conditions. Some example drains are shown in Figure 1.

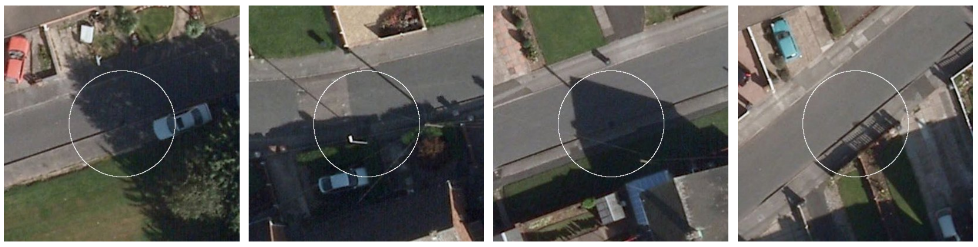
One of the main reasons for incorrect or impossible detection is the case when we have a shadow over our drain, as presented in the first three examples categorized as hard. This is the case in which detection is the hardest mainly because our algorithm is unable to accurately find the road edge and the difference in illumination between the drain and its surroundings is fairly small. Another obstacle would be the shadow made by the pavement, which prevents us from correctly gathering the necessary information to estimate a correct line that defines the road (the method used is described in more detail in Chapter 2.4.1). Furthermore, inconsistencies in the way road lines are painted - or lack of such marking, small protruding shadows or other objects that look like a drain (such as a car's sunroof or dark patches of grass), all make the task of detection more difficult. The methods in which we try to handle each issue and the constraints used are described in detail throughout Chapter 2. In the event that shadows make up a very small portion of the image or are completely missing and the edge of the road is clearly delimited, its shape resembling a straight line as well, the detection is straightforward and 100% accurate as well.



Easy



Medium



Hard

Figure 1: Examples of drains categorized by the difficulty of detection. The ‘Easy’ set presents no shadows or objects that could potentially obstruct detection and the road edge is characterized by a fairly straight line. In the ‘Medium’ set, the drain is still visible, although shadows might be present or the road edge not straight. The drains in the final, ‘Hard’ set are either covered by shadows or are considerably hard to distinguish from their surroundings.

1.1.1 Literature Review

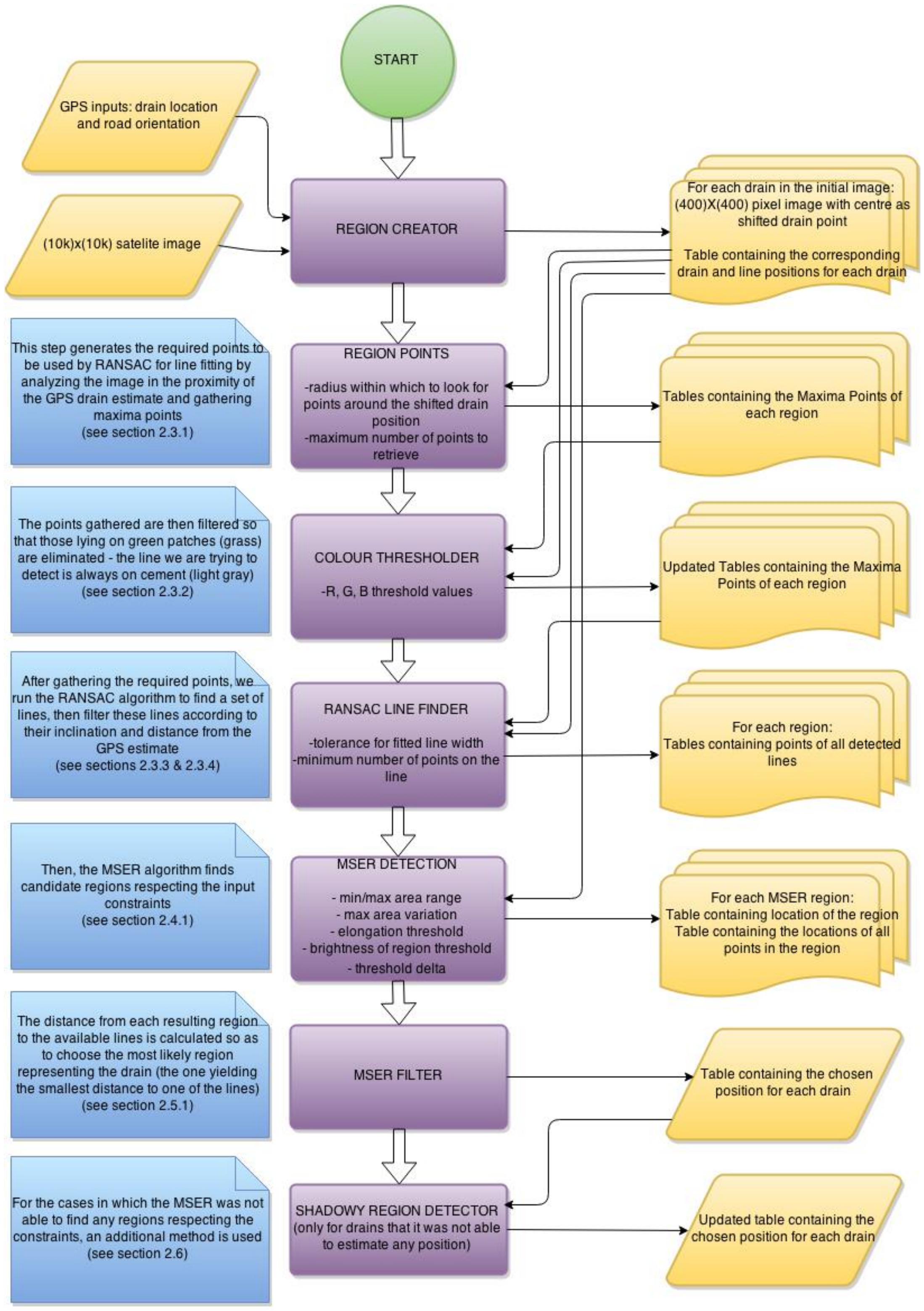
One of the papers available on this topic covers detection of ellipse-shaped manholes using vehicle-based cameras, combined with GPS/INS systems and LIDAR data by illuminating the targets [3] or using mobile light detection alone [7]. While these approaches can give good results (88% detection rate if all the information is used for the first mentioned approach and 97.5% for the latter one), they can prove quite costly. Instead, we would like to only use aerial imagery and GPS data for detection. Even under these circumstances, for an automated system to be used, it would require that the total number of false and missed detections be fairly small. This is the reason for which this task is still done mostly manually. Some research has also been done on detecting round manholes, describing location, size, shape and systematic intensity variations of manhole covers [5]. Our approach however only deals with rectangular ones, found near the edge of the road.

1.1.2 Contributions

The proposed approach worked considerably well on the 'Easy' set (containing 44 drains), achieving an average accuracy of 97.5%. The issues arise when additional elements are present in the scene, shadows being the main cause for missed detections. We will discuss how these can be dealt with throughout Chapter 2. For the set categorized as 'Medium' (34 drains), we obtain an average detection of 81.17%, whereas for the 'Hard' one (23 drains), in which most of the manholes are covered by shadows, we only manage an average percentage of 46.52%.

1.2 Methods Used

The gully drains we are attempting to locate have a consistent appearance and position relative to the road, which makes it relatively straightforward to implement constraints on candidate regions. However, their detection based solely on these constraints is not always possible, due to the complexity of the scene: irregularities in lighting of the region, shadows or physical objects present over the drains or in their proximity, the curvature of the road. The following is a representation of the work-flow carried out for the task of detection. It presents each process, together with the necessary input and the generated output.


START
GPS inputs: drain location and road orientation
(10k)x(10k) satelite image
REGION CREATOR
For each drain in the initial image: (400)X(400) pixel image with centre as shifted drain point
Table containing the corresponding drain and line positions for each drain
This step generates the required points to be used by RANSAC for line fitting by analyzing the image in the proximity of the GPS drain estimate and gathering maxima points (see section 2.3.1)
REGION POINTS
-radius within which to look for points around the shifted drain position
-maximum number of points to retrieve
Tables containing the Maxima Points of each region
The points gathered are then filtered so that those lying on green patches (grass) are eliminated - the line we are trying to detect is always on cement (light gray) (see section 2.3.2)
COLOUR THRESHOLDER
-R, G, B threshold values
Updated Tables containing the Maxima Points of each region
After gathering the required points, we run the RANSAC algorithm to find a set of lines, then filter these lines according to their inclination and distance from the GPS estimate (see sections 2.3.3 & 2.3.4)
RANSAC LINE FINDER
-tolerance for fitted line width
-minimum number of points on the line
For each region: Tables containing points of all detected lines
Then, the MSER algorithm finds candidate regions respecting the input constraints (see section 2.4.1)
MSER DETECTION
- min/max area range
- max area variation
- elongation threshold
- brightness of region threshold
- threshold delta
For each MSER region: Table containing location of the region Table containing the locations of all points in the region
The distance from each resulting region to the available lines is calculated so as to choose the most likely region representing the drain (the one yielding the smallest distance to one of the lines) (see section 2.5.1)
MSER FILTER
Table containing the chosen position for each drain
For the cases in which the MSER was not able to find any regions respecting the constraints, an additional method is used (see section 2.6)
SHADOWY REGION DETECTOR (only for drains that it was not able to estimate any position)
Updated table containing the chosen position for each drain

Chapter 2

Methodology

2.1 Simulating GPS Data

In order to replicate the imperfect data we would have to work with in a real scenario, for each drain, we note its exact position, along with a corresponding point to mark the edge of the road touching it. We may then run the algorithms responsible for shifting the estimated position of the drain and the angle at which the road edge is thought to be by specifying a desired maximal distance from the real position and a desired maximal angle in degrees. The probability of shifting each point by a value of between **-r** and **r** on the X and Y axis (where **r** is selected as 90) and the probability of shifting each angle by a value of between -θ and θ (where θ is selected as 10 degrees) is uniformly distributed. The algorithms will then carry out the displacement for each drain. Figure 2.1 presents an example of the manner in which the GPS coordinates are created by firstly changing the intercept of the line (shifting the two points accordingly) and then changing the edge inclination (by only shifting the second point).

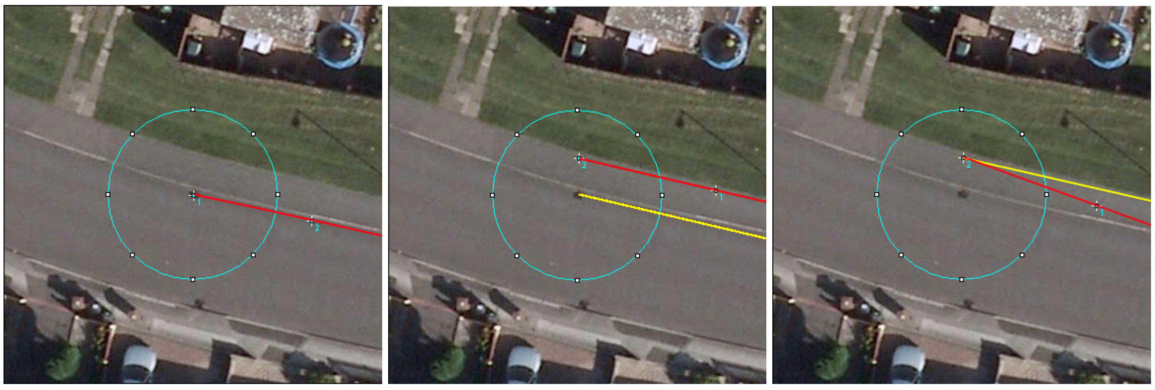

Real positions Drain shifted Angle shifted

Figure 2.1: Simulating GPS data. In the first image, the true line, marked by the two points is represented with the colour red. In the second image, the line has been shifted, keeping the angle consistent. The original line is coloured yellow, while the shifted one is red. In the final image, the angle of the line has been shifted by moving the second point. In the same manner, the previous line is yellow, while the change is highlighted in red.

Before describing the shifting process, we must first introduce some notation: let the X and Y coordinates of the 101 drains in the 10000x10000 image be denoted as $\mathbf{dx_{i=1...101}}$ and $\mathbf{dy_{i=1...101}}$ respectively. Similarly, let us denote the X and Y coordinates of the points used alongside the drain points to mark the edge of the road as $\mathbf{lx_{i=1...101}}$ and $\mathbf{ly_{i=1...101}}$. After inputting the radius within which we wish to shift the points (let us denote it as $\mathbf{r}$), we can proceed with the algorithm. For each value of i, where i ranges from 1 to 101:

1. Generate $\mathbf{n_X}$ and $\mathbf{n_Y}$ ϵ [-$\mathbf{r}$;$\mathbf{r}$] (uniformly distributed), such that $\mathbf{n_X}^2 + \mathbf{n_Y}^2 \leq \mathbf{r}^2$.

2. Create the new, shifted coordinates:

$\mathbf{dx'_i} = \mathbf{dx_i} + \mathbf{n_X}$

$\mathbf{dy'_i} = \mathbf{dx_i} + \mathbf{n_Y}$

$\mathbf{lx'_i} = \mathbf{lx_i} + \mathbf{n_X}$

$\mathbf{ly'_i} = \mathbf{lx_i} + \mathbf{n_Y}$.

We will now simulate the shifting of the line representing the edge of the road by a random angle, lying within an inputted range. We will use the above notation for the coordinates ($\mathbf{dx'_i}$, $\mathbf{dy'_i}$, $\mathbf{lx'_i}$, $\mathbf{ly'_i}$) and note the range value as θ (radians), converting the input (a value expressed in degrees for convenience). For each value of i:

1. Generate $\Delta\theta$ ϵ [-θ;θ] (uniformly distributed).

2. Calculate the slope of the current line: $\mathbf{m_i} = \frac{\mathbf{ly'_i} - \mathbf{dy'_i}}{\mathbf{lx'_i} - \mathbf{dx'_i}}$.

3. Calculate the value of the new angle: $\alpha = \text{atan2}(\mathbf{m}) + \Delta\theta$.

4. Calculate the slope of the new line: $\mathbf{m'_i} = \tan(\alpha)$.

5. Calculate the intercept of the new line: $\mathbf{b'_i} = \mathbf{dy'_i}$ - $\mathbf{m'_i}$ * $\mathbf{dx'_i}$.

The problem now becomes analogous to finding the coordinates of a point (the point that is used -along with the drain location- to represent our shifted line), given that we know the equation of the line on which it lies and the distance between itself and another point on that line (the position of the drain). Let us denote the distance with $\mathbf{d} = 100$ (by convention).

6. Calculate $\mathbf{ly'_i}$ in relation to $\mathbf{lx'_i}$:

$\mathbf{ly'_i} = \mathbf{m'_i} * \mathbf{lx'_i} + \mathbf{b'_i}$.

7. From the length of the line formula, we have:

$\mathbf{d} = \sqrt{(\mathbf{dx'_i} - \mathbf{lx'_i})^2 + (\mathbf{dy'_i} - \mathbf{ly'_i})^2} \Leftrightarrow$

$\mathbf{d}^2 = \mathbf{dx'_i}^2 + \mathbf{lx'_i}^2 - 2 * \mathbf{dx'_i} * \mathbf{lx'_i} + \mathbf{dy'_i}^2 + \mathbf{ly'_i}^2 - 2 * \mathbf{dy'_i} * \mathbf{ly'_i} \Leftrightarrow$

$\mathbf{d}^2 = \mathbf{dx'_i}^2 + \mathbf{lx'_i}^2 - 2 * \mathbf{dx'_i} * \mathbf{lx'_i} + \mathbf{dy'_i}^2 + (\mathbf{m'_i} * \mathbf{lx'_i} + \mathbf{b'_i})^2 - 2 * \mathbf{dy'_i} * (\mathbf{m'_i} * \mathbf{lx'_i} + \mathbf{b'_i}) \Leftrightarrow$
$\mathbf{lx'_i}^2 * (1 + \mathbf{m'_i}^2) + \mathbf{lx'_i} * (2 * \mathbf{m'_i} * \mathbf{b'_i} - 2 * \mathbf{m'_i} * \mathbf{dy'_i} - 2 * \mathbf{dx'_i}) + (\mathbf{dx'_i}^2 + \mathbf{dy'_i}^2 + \mathbf{b'_i}^2 - 2 * \mathbf{dy'_i} * \mathbf{b'_i} - \mathbf{d}^2) = 0.$

For ease of notation, let us now denote:

$\mathbf{bigA} = 1 + \mathbf{m'_i}^2$

$\mathbf{bigB} = 2 * \mathbf{m'_i} * \mathbf{b'_i} - 2 * \mathbf{m'_i} * \mathbf{dy'_i} - 2 * \mathbf{dx'_i}$

$\mathbf{bigC} = \mathbf{dx'_i}^2 + \mathbf{dy'_i}^2 + \mathbf{b'_i}^2 - 2 * \mathbf{dy'_i} * \mathbf{b'_i} - \mathbf{d}^2.$

We now have a polynomial equation of degree 2, from which we need only one of the roots (the other root would correspond to a point on the opposite side of our shifted drain location and would not generate any extra information):

$\mathbf{lx'_i} = \frac{-bigA + \sqrt{bigB^2 - 4 * bigA * bigC}}{2 * bigA}.$

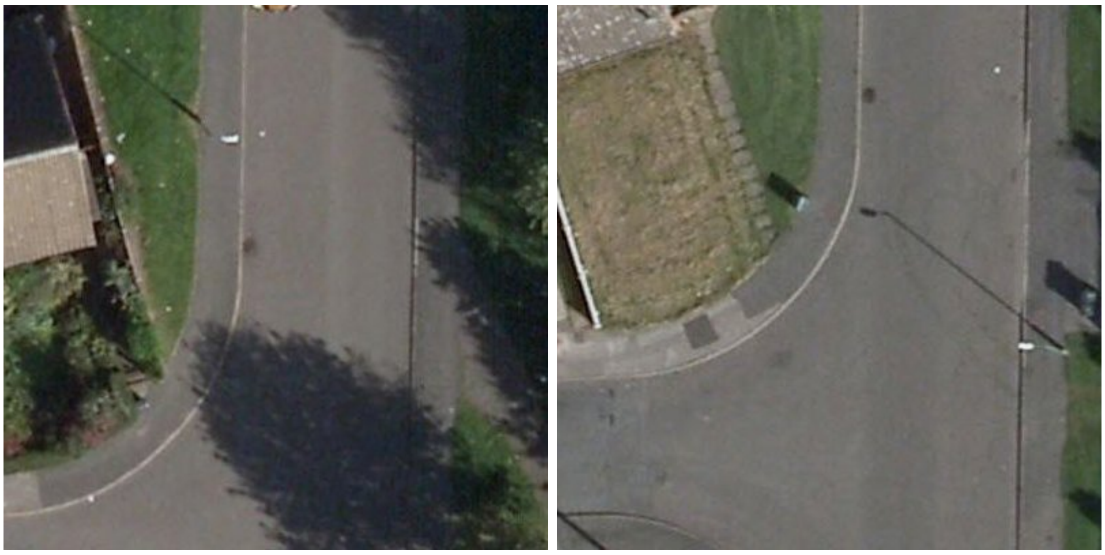
Now, using the formula of the line, we can calculate $\mathbf{ly'_i}$:

$\mathbf{ly'_i} = \mathbf{m'_i} * \mathbf{lx'_i} + \mathbf{b'_i}.$

This will enable us to carry out a detailed analysis of the methods used, while having a variability in the results. Once we have the required knowledge to work with, we can start the process of detection. For the purpose of estimating the accuracy of the process, we will run the randomizing algorithm 20 times and average the results, taking into account the standard deviation as well.

2.2 Creating Region Images and Converting Points

In order for us to easily manipulate the image data, for each of the drains in the 10000x10000 pixel image, we crop a 400x400 pixel region, the center of which being the position of the shifted drain, so that we simulate the inaccurate knowledge. To accompany these images, we also convert the simulated GPS data of the drains and lines to describe the positions relative to our regional images. Thus, the location of the drains will usually be (200, 200), unless the drain is close to the edge of the large figure, in which case our resulting figure will start from the edge as well. Figure 2.2 below presents an example of what a region will usually look like and an example of cropping when the drain is close to the edge of our satellite map.



Normal Region Region Close to the Edge

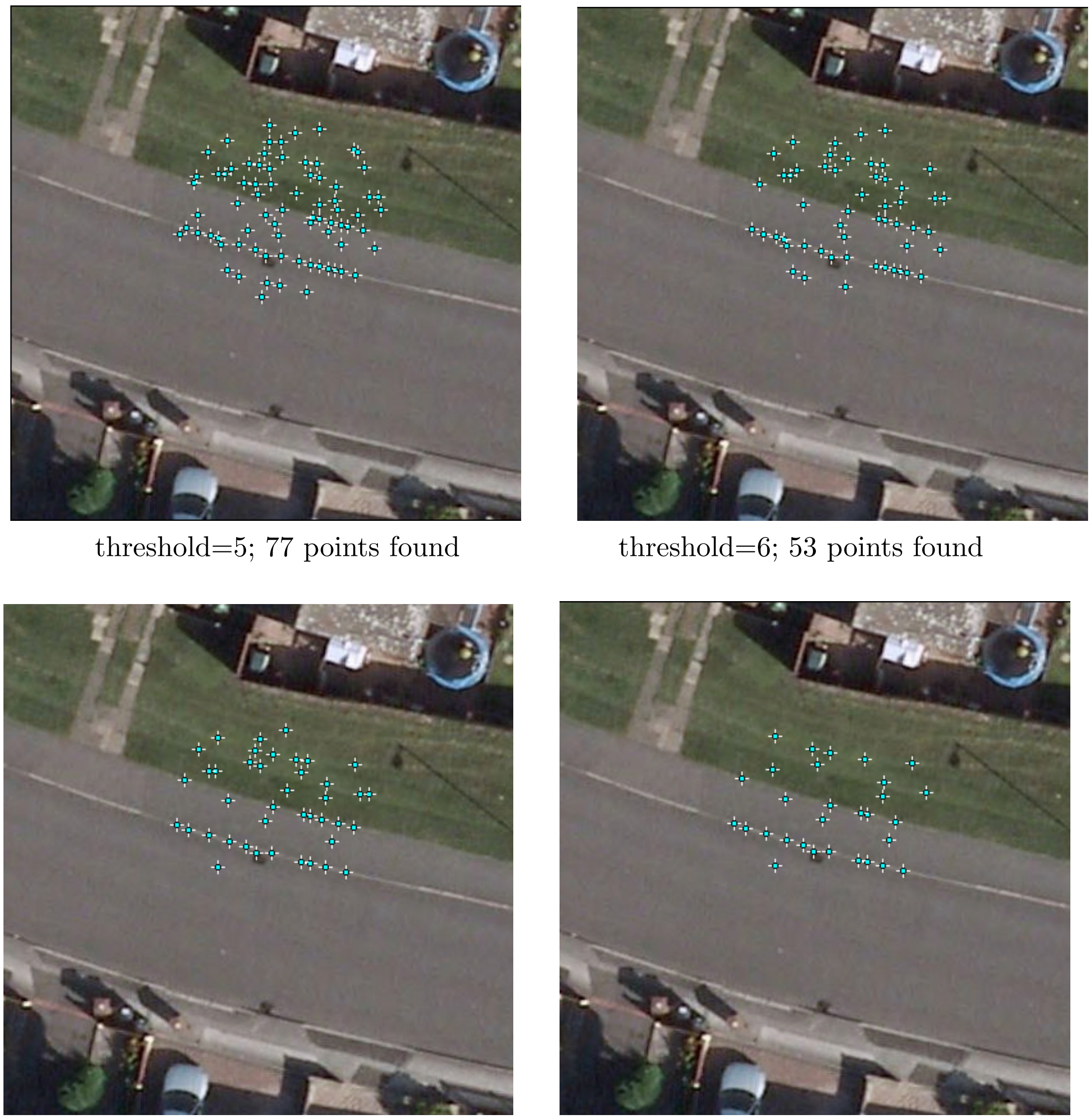
Figure 2.2: Creating Region Images. The first image presents how most regions are cropped (with the drain only slightly deviated from the center, depending on how big the random shifting is). The second image accounts for the case when our drain is located close to the edge of the 10000x10000 pixel image. If this happens, the cropped image used for processing will also have the drain close to the edge, however we keep the image size consistent (400x400 pixels).

2.3 Edge Detection

Due to the fact that 99 out of the 101 drains are located along the pavement line, correctly detecting the pavement line would greatly help us with filtering the candidate regions that are similar to a drain, by choosing the one closest to our detected edge. For this task, we will use the well known RANSAC [1] algorithm, which takes as input a cluster of points, selects a subset of these points that is big enough through which it is possible to fit a line of a specific width, removes them from the initial set and further attempts to find lines using the remaining points.

2.3.1 Maxima Points

The input points for the RANSAC algorithm, responsible for detecting the edge of the road are gathered by calculating the luminance maxima points in the proximity of the shifted gully position and saved in '.xls' files. Maxima are ignored if they do not stand out from the surroundings by more than a certain threshold value, chosen iteratively (the threshold is increased until the number of maxima points is below a certain inputted level). For this task, we use ImageJ's 'Find Maxima' command [6]. For example outputs of iterative threshold choosing based on a maximum number of 30 points, see Figure 2.3 below.



threshold=5; 77 points found

threshold=6; 53 points found

threshold=7; 41 points found

threshold=8; 30 points found

Figure 2.3: Choosing the correct threshold. The images above present the points generated by the 'Find Maxima' command, using different threshold values. The algorithm starts with an initial threshold value of 1 and continually increases it until the generated number of points is smaller than or equal to the inputted stop value. If the stop value would have been 30, for the image presented here, the algorithm would stop after threshold=8 and would save the newly generated points.

2.3.2 Filtering using Colour Intensity

Once the appropriate amount of input points has been gathered, we use a colour intensity filter to remove the points on entities that are probably not part of the road such as grass or cars. This is achieved by extracting the three RGB channels using ImageJ's function 'RGB stack' [8], converting them to binary masks and applying a minimal and maximal threshold. An example of the three colour channels and of filtering using a minimal intensity threshold of 100 and a maximal one of 220 for all three channels can be seen below in Figure 2.4.

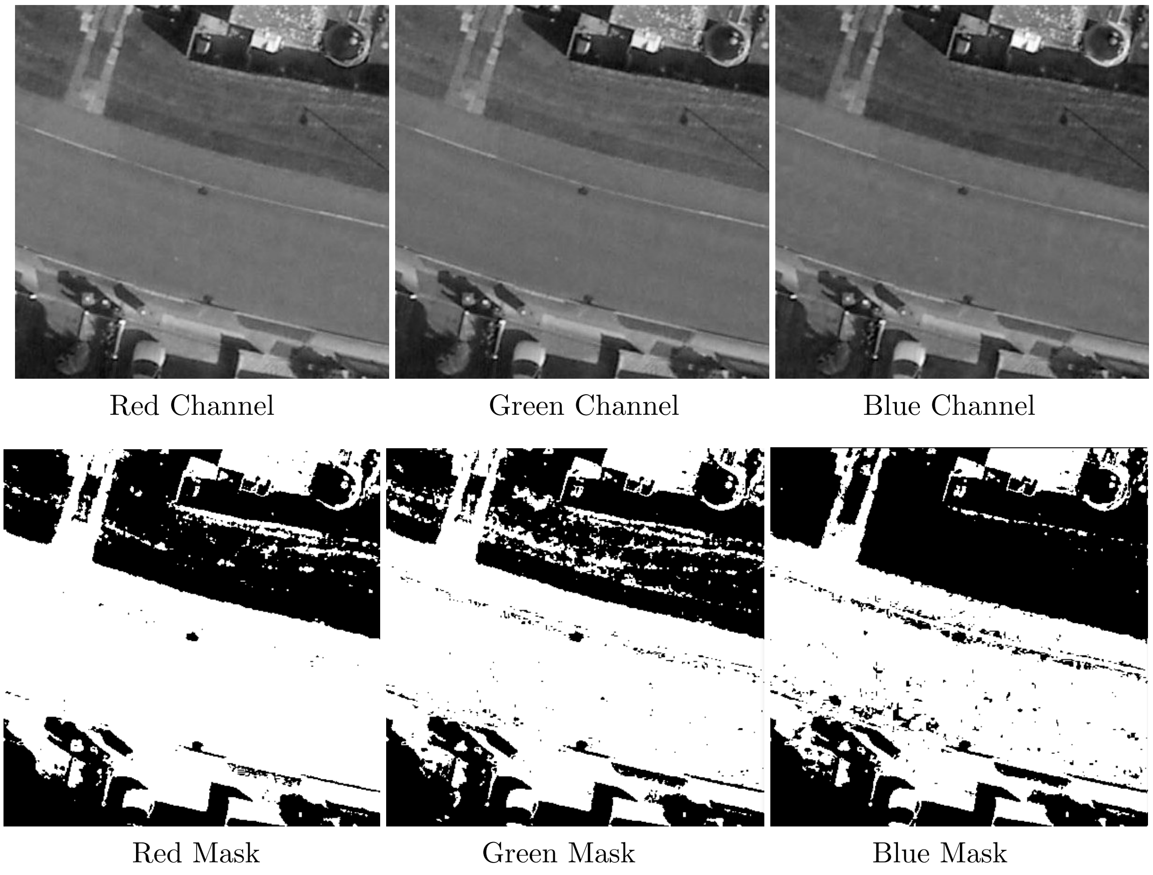


Figure 2.4: Thresholded Channels. The first three images are the extracted Red, Green and Blue channels of the image. The last 3 images are the masks generated after applying a minimal intensity threshold of 100 and a maximal one of 220 to each of the three channels.

After filtering the three separate channels, they are put together so that only pixels respecting the thresholds in all red, green and blue images are represented. In order for us to eliminate very small, isolated regions that are accepted and rejected, morphological opening and respectively closing of the image is performed, as shown below in Figure 2.5.

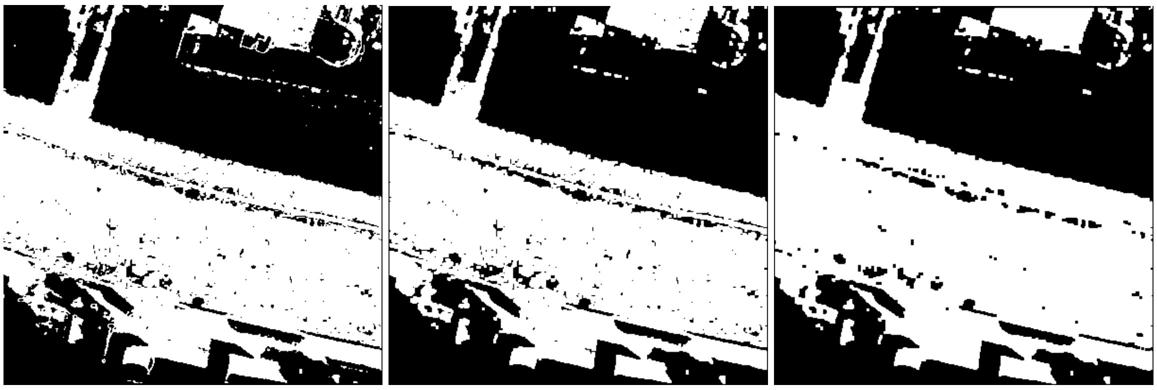


Figure 2.5: Morphological Opening and Closing. The first image is the output of 'putting together' the three masks presented earlier. Only pixels respecting the thresholds in all 3 masks are accepted (white). The second image is the result of performing morphological opening on the first image, while the third one is the result of morphological closing performed on the opened image.

In some cases, due to the elevation of the pavement at the edge of the road and the orientation of the sun, a shadow might be formed next to the road edge. This can potentially result in 'covering up' of the line we want to detect after the morphological operations are performed. We try to counter this by dilating the image, as presented in Figure 2.6.

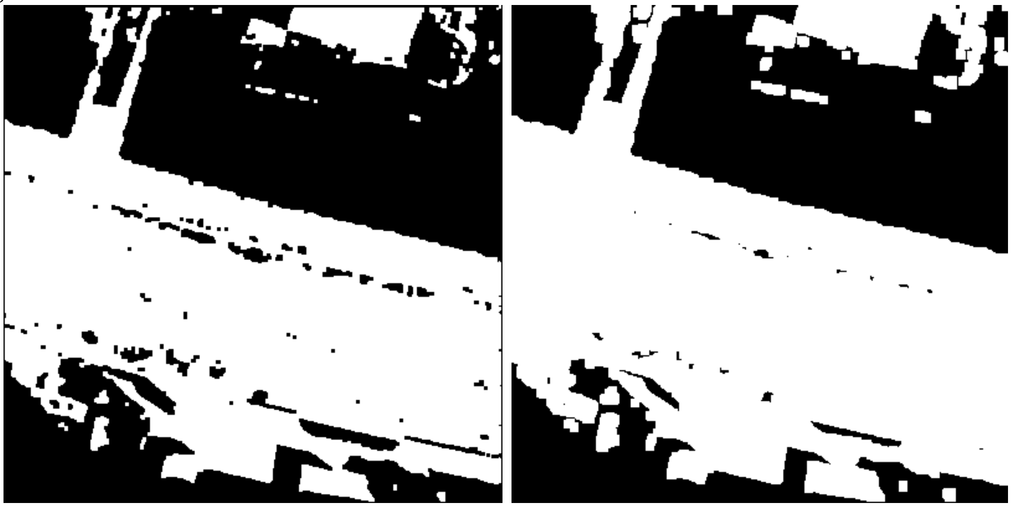


Figure 2.6: Dilating Image. The second image presents the result of morphological dilation on the already opened and closed mask. This step is performed so as to ensure that points lying on the shadow created by the pavement are also taken into consideration.

The next step in the process is eliminating the Maxima Points that do not respect the constraints. This achieved by entering the '.xls' file in which we save the points and removing the ones which are covered by the binary mask created below in Figure 2.7.

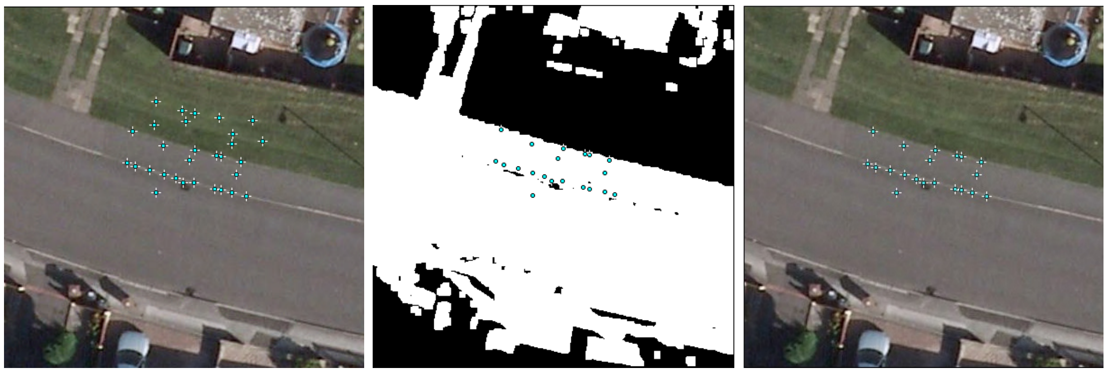


Figure 2.7: Filtering Points. This figure shows how the maxima points are filtered using the generated mask, thus eliminating those situated on the green patches. To be more exact, if the position of the maxima point coincides with a white pixel in the mask, we accept it; otherwise, we delete it from the file corresponding to this region. This process will make the RANSAC line finding algorithm more accurate, while also lowering the number of false line detections.

2.3.3 RANSAC Line Fitting

After filtering the points, we can proceed to use them with RANSAC, while also specifying the appropriate parameters: the width tolerance of the fitted line (**tol**) and the minimum amount of points that the line must contain (**MINLEN**). For an example output of the algorithm with parameters of 3 pixels in width and 7 points, see Figure 2.8.

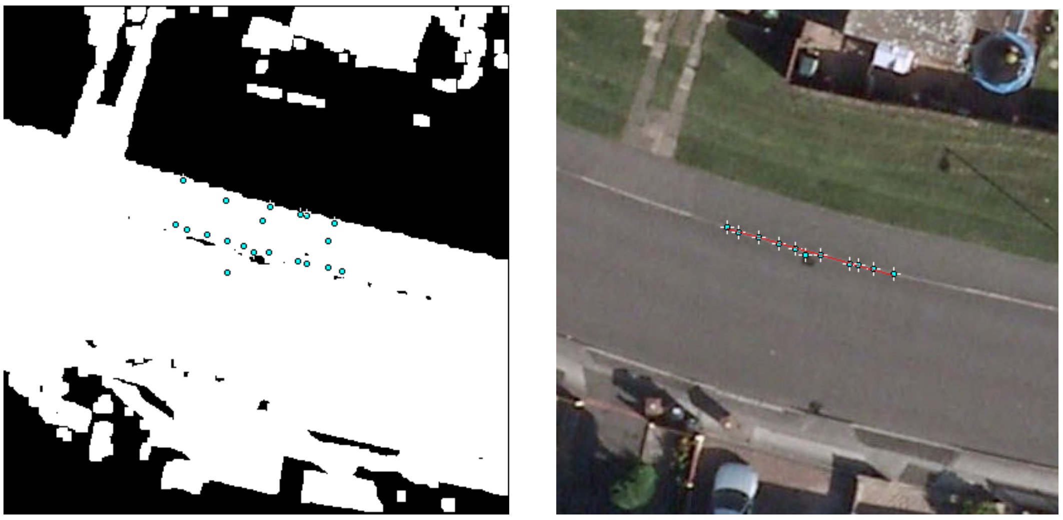


Figure 2.8: Result Returned by RANSAC. After the maxima points have been gathered and filtered using the mask presented in the first image, we apply RANSAC to find the candidate lines. In this case, due to the fact that the minimum amount of points that must be contained by a fitted line is 7, after fitting the line presented and eliminating the used points from the entire set, no other line can be found.

For each drain, we access the '.xls' file in which the maxima points are stored and apply RANSAC to them. From the given set, we select two different random points and check which other points from the set are within the width tolerance threshold from the line created by the randomly selected locations. If the total number of points that respect this constraint is smaller than our inputted threshold for the minimal number of points, we select a different random pair, until the constraints are met. If this does not happen after N tries, we stop and assume no line can be found. This value of **N** is calculated by using the formula $\mathbf{N} = \frac{log(\mathbf{probf})}{log(1-\mathbf{probd}*\mathbf{probnd})}$, where **probd** is defined as the probability of a pixel being part of a line, **probnd** is the probability that the second pixel is on the same line and **probf** is the allowed failure probability. Using the observations of the output, we assume that the algorithm finds an average of 3 lines, each line containing on average 7 pixels. For example, if we are using around 70 points as an initial set (a value that gives good results), then **probd** = 0.3 and **probnd** = 0.1. If we use a reasonably low allowed failure probability of **probf** = 0.01, the value of N rounded to the closest integer will be 66.

Returning to our initial problem, after finding a subset of points through which we can fit a line of width **tol**, we store them and continue applying RANSAC on the remaining points in the set, until all points have been used or the algorithm is unable to find another line. At the end of the algorithm run, we end up with a few candidate lines (in most cases between 2 and 5).

2.3.4 Line filtering

After gathering the candidate lines, we use the simulated GPS line to decide which one is most likely the correct road edge. We do this by first getting an equation of the GPS line in terms of $\cos(\theta)$ and $\sin(\theta)$. For each drain, while iterating through the sets of points used to represent the candidate lines, we fit a least squares line through them and obtain its equation as well, using Matlab's 'polyfit' function. Thus, let **b** represent the value of the candidate line's slope in radians. Let us now calculate $\Delta\theta$ as the difference between the two lines' inclination: $\Delta\theta = \text{rad2deg}(atan2(\sin(\theta), \cos(\theta)) - atan(|b|))$ in degrees (for easier comprehension). If this value is greater than an inputted threshold (somewhere between 10 and 15 degrees gives the best results), then we do not take this line into consideration.

If our line passes this initial stage of filtering, we proceed to calculate the mean distance from a point on our candidate line to the GPS line. For this purpose, let us denote $\mathbf{lx_i}$ and $\mathbf{ly_i}$ the X and Y coordinates of the points on our line, where $i\epsilon 1...M$, where M is the total number of points on the line. Now, for each of the M points:

1. Calculate the slope of the perpendicular to the GPS line $\mathbf{m_2}$ and the corresponding intercept $\mathbf{inter_2}$:

$$\mathbf{m_2} = \frac{-1}{atan2(\sin(\theta), \cos(\theta))}$$

$$\mathbf{inter_2} = \mathbf{ly_i} - \mathbf{m_2} * \mathbf{lx_i}$$

2. Find the X and Y coordinates of the point of intersection between the two lines $\mathbf{Ix}$ and $\mathbf{Iy}$:

$$\mathbf{Ix} = \frac{\mathbf{inter_2} - \mathbf{inter}}{\mathbf{m} - \mathbf{m_2}}$$

$$\mathbf{Iy} = \mathbf{m} * \mathbf{Ix} + \mathbf{inter}$$

3. Calculate the distance between the intersection point and the point on the candidate line:

$$\mathbf{distance_i} = \sqrt{(\mathbf{lx_i} - \mathbf{Ix})^2 + (\mathbf{ly_i} - \mathbf{Iy})^2}$$

After performing the calculations for each point, we calculate the average distance:

$$\mathbf{distance} = \frac{\sum distance_{\text{i}}}{M}$$

After calculating the average distance for each candidate, we keep only those lines whose average distance from the simulated GPS line is smaller than 140 pixels (this value is considerably large to avoid losing the true line in this process, while also ensuring we are not storing the line marking the opposite edge of the road).

2.4 Gully Drain Candidate Regions

2.4.1 MSER Region Detection

The purpose of finding the estimated location of the road edge is to use it to check whether our assumptions for the drain locations are correct and decide which of the candidates is the correct one. To arrive at this stage, we need to find regions similar to a drain.
The first step in this process is using the well known MSER algorithm (more precisely, Matlab's 'detectMSERFeatures' implementation [9]) with a greyscale image of an area of interest, chosen by cropping a rectangle of width and height of 200 pixels corresponding to the diameter of the circle used to simulate the inaccurate GPS coordinate of the drain, the center of which being the shifted drain point. Figure 2.9 presents an example of raw output given the whole region image (400x400 pixels), with no constraints.

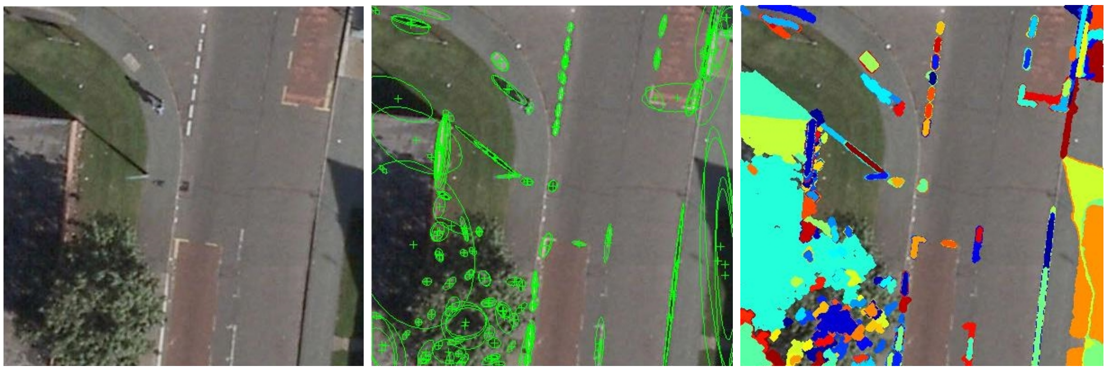


Figure 2.9: Result Returned by MSER. This is an example of output using Matlab's 'detectMSERFeatures' on the whole image, with the default parameters. The outputs of the second and third image are the same, only the visualisation method differs, in the second image, the regions being represented by ellipses, while in the third, each region's pixels are coloured with a specific colour.

The regions detected are filtered according to what size range is allowed (achieved by detectMSERFeatures's 'RegionAreaRange' parameter - a vector defining what the accepted minimal and maximal area values are: [**minAreaRange**, **maxAreaRange**]=[45,95]) and checked for stability (using the **thresholdDelta**=2.1 and **maxAreaVariation**=1.5). The values for these parameters are further explained in Chapter 3.3.1. After this step, they are filtered according to their shape; If they are too elongated (if the ratio between the longer and shorter axes of a fitted ellipse is greater than 4), they are excluded.

A considerable problem in this situation is the fact that the algorithm will often pick up white markings on the road, some possibly being quite close to our drain (e.g. markings for bus stops). To address this issue, we also remove the regions based on the mean colour intensity of the region. If the MSER region is brighter than the surrounding image, it cannot be a drain. The implementation of this constraint consists of looping through each pixel of the candidate region and cumulating their intensity values. At the end of this step, we divide by the total number of pixels in the region. From this we obtain an average intensity value. If this value is above 100, then we may take it into consideration. Figure 2.10 provides an example of filtering by region intensity.

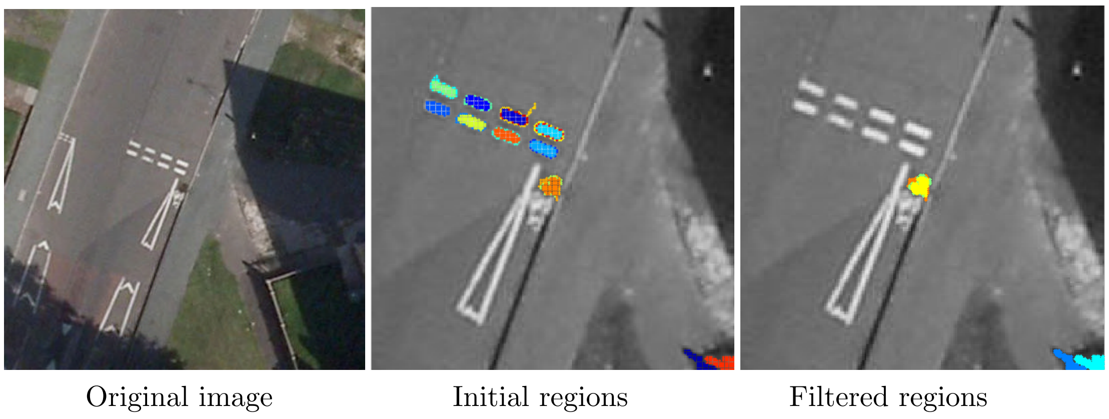
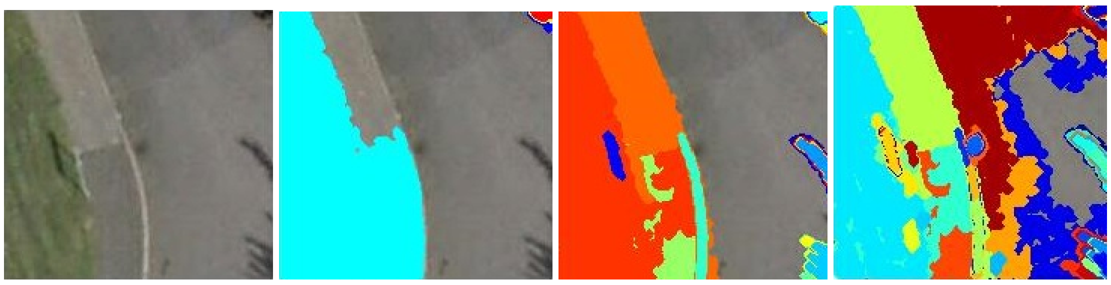


Figure 2.10: Initial result of MSER and Result After Filtering. Left: Original image. Center: Results of MSER algorithm without intensity threhsolding. Right: Results after applying intensity threshold. The remaining regions present in the lower-right corner will be later filtered, since their distance to the candidate lines are greater than the regions representing the drain (present in the center of the image).

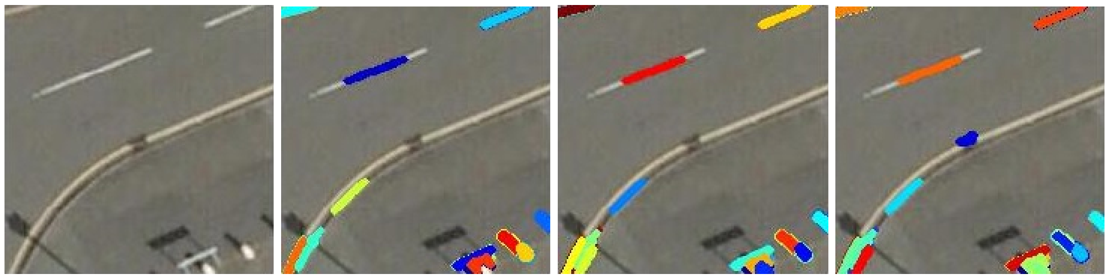
The process of MSER detection carried out above starts from the inputted parameters of the detectMSERFeatures: **minAreaRange** = 56, **maxAreaRange** = 100, **ThresholdDelta** = 2, **maxAreaVariation** = 1, while also using the previously mentioned filters to output a set of candidate regions. If, however, this run does not return any results, we decrease the value of **ThresholdDelta** by 0.1 and re-run the analysis, in effect decreasing the step size between the intensity threshold levels used to select extremal regions, thus allowing us to retrieve sections where the difference of intensity between the inner-pixels and the extremities is more obscure. The process of lowering **ThresholdDelta** is repeated until we have at least one region returned or **ThresholdDelta** reaches the 0.8 mark. Figure 2.11 presents an example of different results of detectMSERFeatures by varying this parameter and using the default values for the remaining ones.



Original region thresholdDelta=4 thresholdDelta=2 thresholdDelta=1

Figure 2.11: Varying thresholdDelta. Results outputted by the MSER algorithm with different values for the thresholdDelta parameter. If the difference between the drain's pixels and it's surroundings is fairly small, lowering this value will help us find the drain, as shown in the forth image, using thresholdDelta=1.

In the event that the algorithm has still not returned any regions, in the same manner, we begin to iteratively decrease the value of **MinAreaRange** by 5 and re-run the algorithm until we have at least one region or the value drops as low as 34. See Figure 2.12 for an example output of detectMSERFeatures with different values for **MinAreaRange**.



Original region minAreaRange=54 minAreaRange=44 minAreaRange=34

Figure 2.12: Varying minAreaRange. Results outputted by the MSER algorithm using different values for the minAreaRange parameter. In some cases, due to the fact that a line is painted over our drain, we need to decrease the minimal area accepted, so as to at least detect half of our drain, as shown in the fourth figure using minAreaRange=34.

In some cases, even after lowering both these MSER parameters, we are unable to find regions that also match our shape and average intensity criteria. This is when we run our detection one last time, this time using a slightly different image as input. Apart from the case when a shadow is covering our drain (we will discuss how to deal with this in Chapter 2.6), a common reason for the inability of detecting suitable regions is the shadow cast by the pavement, 'linking' the possible region with the shadow, thus contrasting extremity points being lost. To account for this, we will use a method called gray-level morphology to fill in the section of the shadow with a gray colour, similar to that of the road.

Firstly, we calculate the angle (let us call it α) of the line perpendicular to our approximated GPS line (determined by two points: [**dx**;**dy**] and [**lx**;**ly**]) used to mark the edge of the road:

α = atan2(**lx-dx**, **dy-ly**) * 180 / pi + 90;

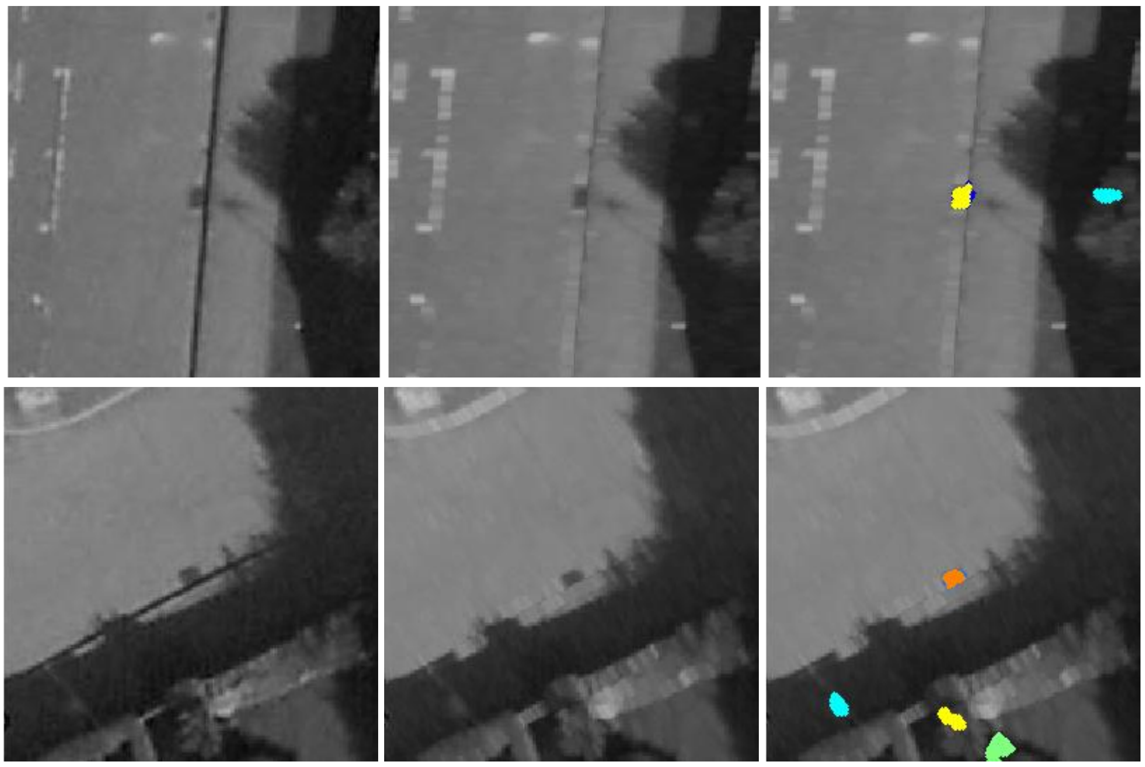
After this, we use Matlab's 'strel'[10] function, responsible for creating a morphological structuring element-in our case a line having a width of 6 pixels (this being roughly the width of the shadow) at an angle α-:

SE = strel('line', 6, α);

We use this structuring element to dilate the grayscale image of the area we are analyzing and fill in any elements that resemble a line parallel to our GPS edge.

grayROI = imdilate(grayROI, SE);

Despite the fact that it creates a slightly blurry image, the results are generally useful, provided that the estimated angle is close to the real value and the drain isn't connected to any other shadows (as seen below in the third example of Figure 2.13).



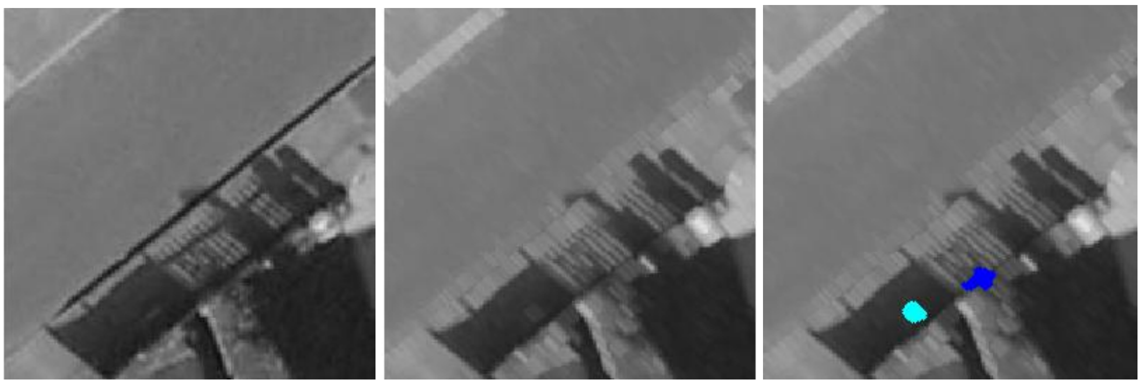


Figure 2.13: Performing Gray Morphology. In most cases, the dilation performed is enough for us to detect our drain, as can be seen in the first two examples. Due to the fact that some of the regions are also distorted, we detect some false positives as well, which will eventually be filtered. Generally speaking, if our drain is within the candidate regions, the filtering algorithm described in Chapter 2.5 does a good job of selecting the correct area. In some cases, the dilation might be enough to fill in the line-shaped shadow, however the drain might still be connected to another dark region, in which case the algorithm will not work as desired.

2.5 Linking RANSAC and MSER

2.5.1 Calculating distance

In order for us to choose which region is most likely to be the drain, we calculate the distance from each remaining region to the remaining detected lines. The smallest distance will be taken to correspond to the gully, as they tend to be placed next to the road.

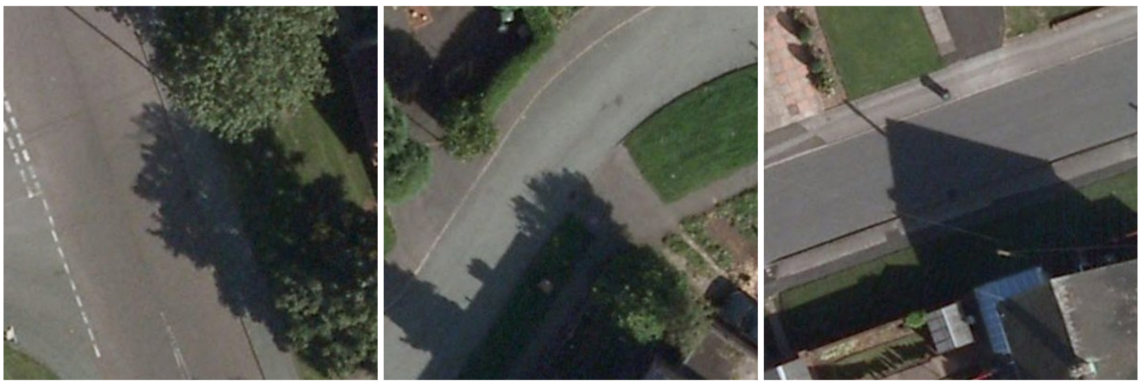
2.5.2 Verifying Against True Positions

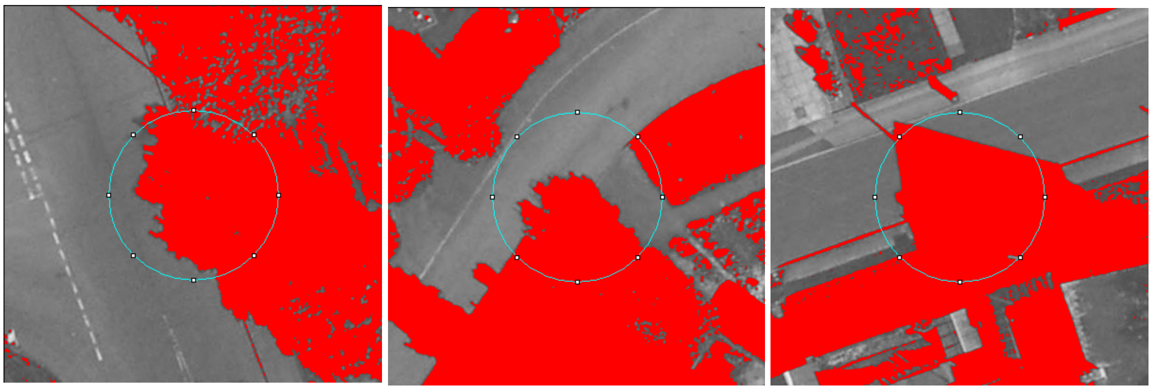
Each prediction is then compared to the real position and if the error is smaller than 15 pixels, we consider it a correct estimate.

For statistical analysis, in the case of all 101 drains that we are looking for, we save a detailed excel file of each drain position estimate, or the reason behind the lack of an estimate. This may be caused by the inability of either algorithms to detect a line or a region with the specified constraints. Furthermore, a summary table is generated for each run of the whole process, containing the value of all parameters used and the accuracy of detection (percentage of drains correctly identified, wrongly identified or unable to identify). This will later help us in analyzing the constraints and deciding the optimal ones by plotting and reviewing ROC curves.

2.6 Analyzing Remaining Regions

For the cases in which the MSER algorithm is unable to find any regions, we try to use a different approach, based on the general lighting intensity of the area in the estimated GPS position's proximity. Due to the fact that the most common cause of lack of detection is the presence of a shadow over our drain (making the contrast between our gully and its surroundings very low), we will try to detect when this is the case and attempt detection by thresholding with a very low value (39) and applying despeckle. We decide that a shadow does exist by performing auto-thresholding on the image region within a circle of radius 100 and center as the GPS position using ImageJ's 'setAutoThreshold("Default")' method after converting it to an 8-bit type and looking at the total area of the created binary mask. If more than 80% of the image appears to be dark, we proceed with thresholding using the small value. For an example output, see the Figure below.


Original Images


Auto-thresholding performed

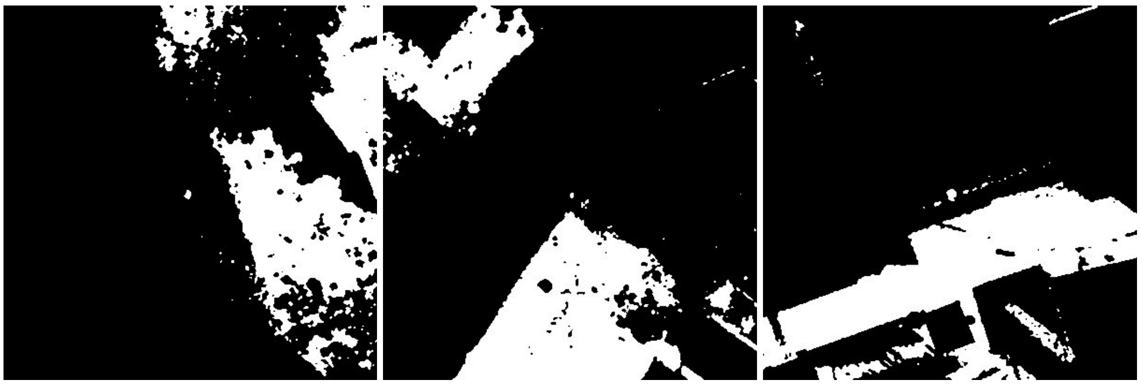

Thresholded and despeckled

Figure 2.14: Detecting drains covered by shadows. The upper three images are examples of images for which the MSER algorithm does not manage to find any MSER regions respecting the specified constraints. We now want to check whether the drains are covered by shadows. The middle three images present the result of auto-thresholding, which usually proves to be a good method of returning the amount of shadow present. After checking the proportion of shadow present within a circle of radius=100 pixels in the GPS drain position's proximity, if this value is over 80%, we proceed with thresholding the original image using a value of 39 and despeckling. The results can be seen in the bottom three images of hte figure.

From this output, we look at all the objects within the circle, and return the object with the largest area, excluding those that extend as far as the margin of the image (so as to filter the dark regions of the ground or larger objects).

If however the proportion of the dark regions is smaller than 80%, we assume that a small shadow made by the pavement is present - this being the second most common reason for missed detections, however the gray-level morphology attempted as part of the MSER detection did not do a good enough job of eliminating it (this can happen when the estimates for the GPS road edge are inaccurate, thus resulting in a blurred image, with the shadow still being connected to our drain). In this situation, the algorithm performs auto-thresholding, erosion to eliminate the pavement shadow, despeckling to clean the image of small particles and dilation to account for the erosion previously performed. The resulting objects are filtered so that their circularity is greater than 0.5 and the object with the greatest area is selected as the correct estimate. Figure 2.15 below presents the results of the steps described.

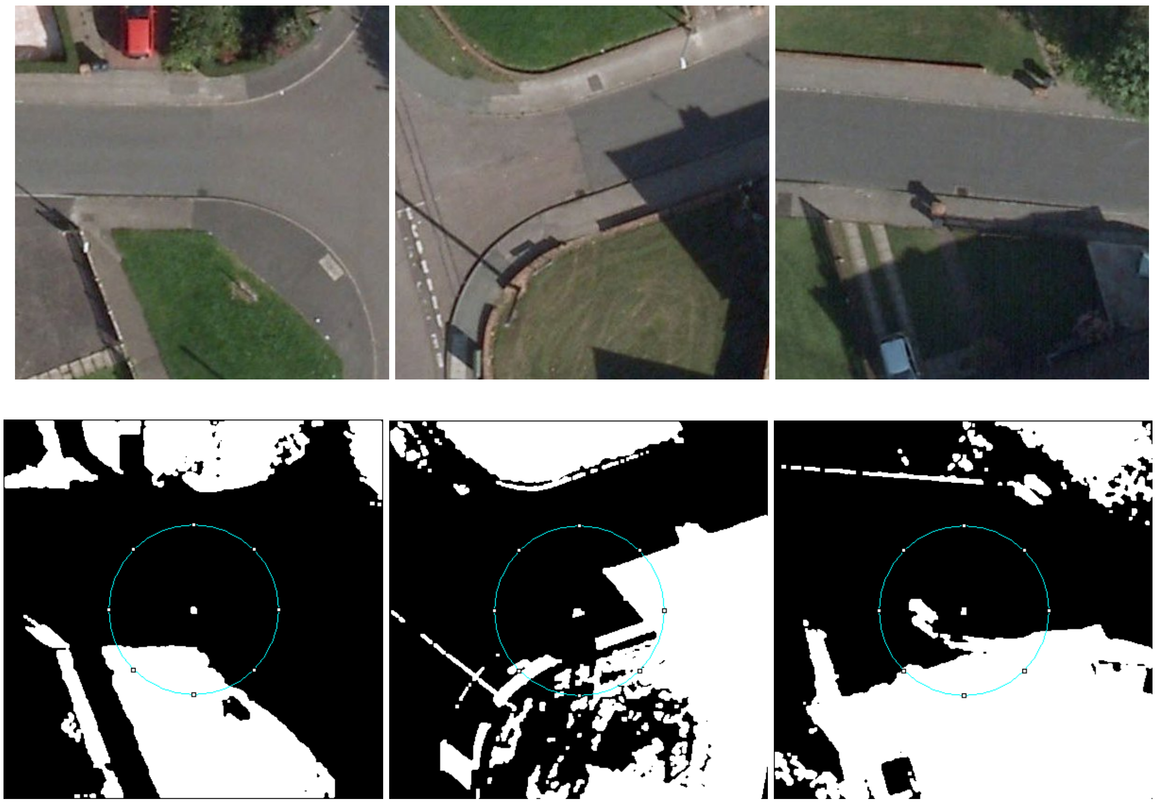


Figure 2.15: Attempting to eliminate shadow. For the first three images, the MSER algorithm was also unable to find regions conforming to the specified constraints. After the auto-thresholding was performed however, the amount of shadows found was under 80%, thus the second methodology was used. The bottom three images present the result of auto-thresholding the images, performing the required morphological operations: erosion, despeckling and dilation.

Chapter 3

Experiments

3.1 Experimental Procedure

In order for us to evaluate the performance of our proposed workflow, besides looking at the percentages of correct and wrong detections, we would like to study its sensitivity to parameter changes and also find the optimal parameters for our set of data. We will analyze the results of the whole workflow, as well as look at the performance of the two major parts of the process: road edge detection and MSER region detection. For optimization, we will carry out an elitist type of genetic algorithm, where we gradually 'mutate' each variable and update it according to which values of that variable give the best solutions.

An approach to getting the optimal parameters with 100% accuracy would be to run the algorithms for all possible combinations, plot a solution space representing the number of true positives and false positives and choose the optimal solution from the whole space. Despite this being the most accurate method, due to time constraints, we will use the method mentioned below.

To account for differences in the inaccuracy of GPS estimates, we simulate this by shifting both the GPS approximation of the drain position and that of the angle at which the road lies in the image.

The algorithms are run multiple times, with varying parameters, the results being used to plot performance curves for ease of visualization and analysis.

3.1.1 Data

The data available for this project is a 10000x10000 pixel satellite image of streets in Edinburgh. Each location of a drain is tagged using a single XY coordinate within the image, along with a secondary XY coordinate to give the inclination of the road edge close to the drain (as explained in Chapter 2). Each drain has been categorized into one of three test sets (Easy, Medium and Hard) according to their difficulty of detection, as mentioned in Chapter 1.1, in the following proportion:

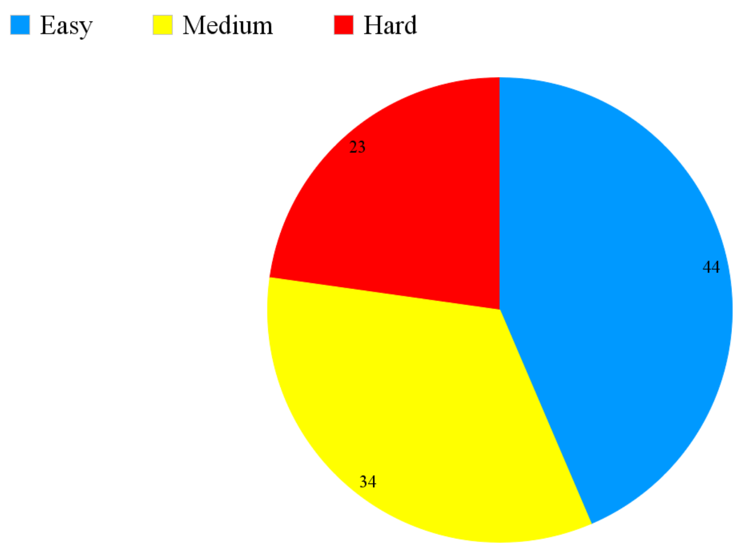


Figure 3.1: Categorizing the Drains. This chart presents the proportion of drains categorized as ‘Easy’, ‘Medium’ and ‘Hard’ out of the total 101 available.

3.2 Edge detection

For analyzing the RANSAC algorithm, we must look at the parameters used, namely: the tolerance for the width of the fitted line (**tol**), the minimal number of points required to be found on the line (**MINLEN**) and the maximum number of maxima points generated for each image to use with RANSAC (**NP**). By selecting a very small initial value of **tol**=2 for the tolerance and a value observed to yield good results for **NP**=70, we run the algorithm multiple times, varying **MINLEN**. We can then look at the how the number of cases in which we are able to extract at least one line and the total number of cases in which we obtain the correct line varies depending on the value of **MINLEN**.

From Figure 3.2, we can observe how the number of correct identifications of the true line is fairly stable over the range 3-9, with the peak being achieved at the value of **MINLEN**=6, the number of false lines increasing as the threshold is lowered. Our first priority when optimizing will be to increase the number of true positives; if multiple parameters happen to achieve this, we take the one which minimizes the amount of times we end up with only false negatives (the difference between the two values presented).

Let us now carry out the same type of run, this time using **tol**=6, **NP**=70 and varying **MINLEN**. From the second figure below, we can see that we obtain the most true positives with a value of **tol**=4, improving the detection from 82 to 84 out of 101. This value also coincides with a lower number of false negatives, as indicated by the green line.

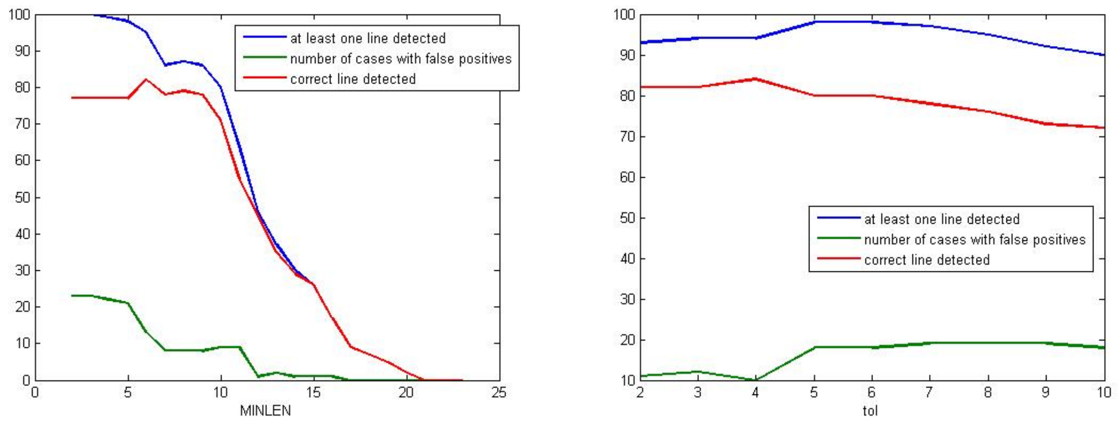


Figure 3.2: Number of cases true line found (red), number of cases at least one line found (blue), number of cases false positives found (green) when altering MINLEN & using tol=2 and altering tol and using MINLEN=6.

Using the new value of **tol**, we must now re-optimize **MINLEN**, in effect repeating the first step. This will converge to the same, optimal result: **tol**=4, **MINLEN**=6.

We can now test **NP** for optimality using the parameters above. As shown in Figure 3.3, the results are most stable and close to optimality when we are looking to generate more than 60 points. In order for us to reduce the number of false positives and to use a 'safe' value within the optimal range, we conclude that **NP**=70 is an adequate value to use.

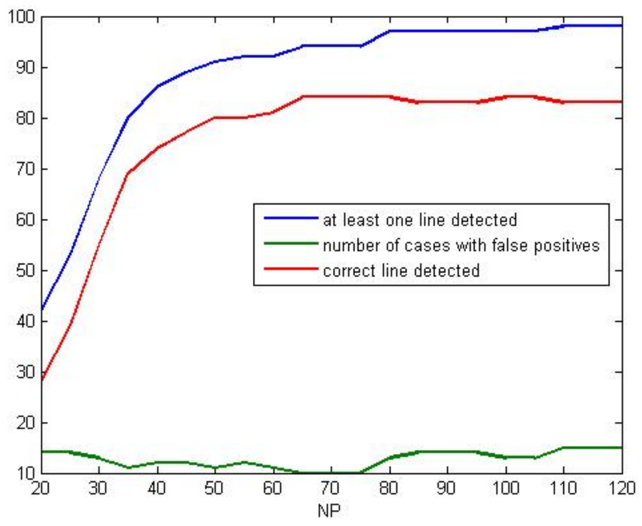


Figure 3.3: Results of Varying **NP**

3.3 Drain detection

3.3.1 MSER Algorithm

To evaluate the performance of the Drain detecting algorithm, we will look at the parameters used by MSER (i.e. **ThresholdDelta**, **maxAreaVariation**, **minAreaRange**, **maxAreaRange**, **maxAreaVariation**) and at the line width used for performing gray-level morphology separately (let us call it **glmWidth**).

In spite of the fact that the drain-finding algorithm iteratively lowers **threshhold-Delta** and **minAreaRange**, we must still decide what to initialize them with, alongside **maxAreaRange**, **maxAreaVariation** and **gmlWidth**.

Since the parameter hardest to estimate is **ThresholdDelta** (whose value typically ranges between 0.8 and 4), we will choose it as the first value to optimize, keeping the rest stable.

Using the knowledge that we already have of the algorithm outputs, we estimate that the following would be good values for the remaining parameters:
minAreaRange=50;
maxAreaRange=100;
gmWidth=6;
maxAreaVariation=1.

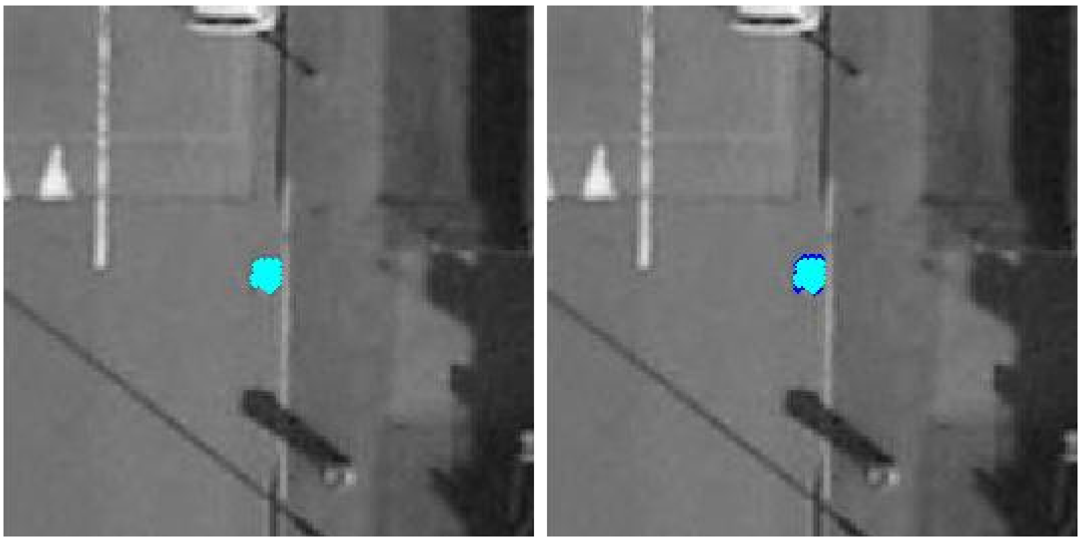
As we vary **thresholdDelta**, we observe that the results are quite stable along the whole range, with the optimal results being obtained in the subset **thresholdDelta**ϵ[2, 2.2]. Within this subset, we manage to detect the correct position for our drain in 79 of the 100 cases using this approach, as shown in Figure 3.5. Let us choose the median, 2.1 as the correct value and proceed with varying **minAreaRange**, responsible for filtering regions that are too small.

In the case of **minAreaRange**, we notice that using a value of over 70 for this parameter will considerably affect our results, however the number of true positives is fairly constant throughout the runs where this parameter is set to under 70, with the optimal results (correct 79 detections) being registered with a value of below 50 (see Figure 3.5). Since this algorithm will return all regions that satisfy the constraints, we must ensure that the number of erroneous regions returned is minimized by choosing a value in the upper region of the [0, 50] range. We will account for possible cases where a drain is slightly smaller than the ones we have worked with for this project by selecting **minAreaRange** = 45 as the optimal value.

We will now proceed to optimize **maxAreaRange**, responsible for filtering regions of too big areas. Similarly, we may observe that we obtain optimal and constant results by using a value greater than 90 (79 drains correctly identified as possible candidates).

Applying the same methodology as above, we would like to minimize the number of erroneous regions returned, while still leaving way for extreme cases. We will thus select 95 as the optimal value for this parameter.

As far as **maxAreaVariation** is concerned, this parameter used by the Matlab implementation of the algorithm typically ranges from 0.1 to 1; the higher the number, the more MSER regions are returned stemming from the same location, but varying in stability. As exemplified in Figure 3.5, a high value does not influence our algorithm's accuracy, since the less stable regions have roughly the same location, the ones that are too big being filtered by **maxAreaRange**. Thus, the reason for which we are using a high value of 1.5 for **maxAreaVariation** is because it ensures we are able to find our drain at no cost other than having to store a few extra regions and it is reasonably within the set [1, 2], for which we obtain the optimal result, respectively 79 drains correctly identified as candidates.



maxAreaVar=0.1; 1 region   maxAreaVar=1; 2 regions

Figure 3.4: Results with different values of maxAreaVariation. These two images show the output of the MSER algorithm using 0.1 and 1 as values for the maxAreaVar parameter. When using a greater value, more regions might be returned at the cost of their stability. Opting for a greater value is a safe option, since it increases our chances of finding drains with a slightly more unstable pixel region.

When analyzing **gmWidth** (responsible for specifying the line width of the structuring element used for our gray dilation), we only look at the 9 cases in which the MSER algorithm is not able to find any regions satisfying the specified constraints. As can be seen in Figure 3.5, we can obtain good results by modeling a line of width of 3 to 8 pixels, with the optimal results achieved using **gmWidth**=5 or **gmWidth**=6, correctly identifying 7 out of the 9 regions after dilating the image.

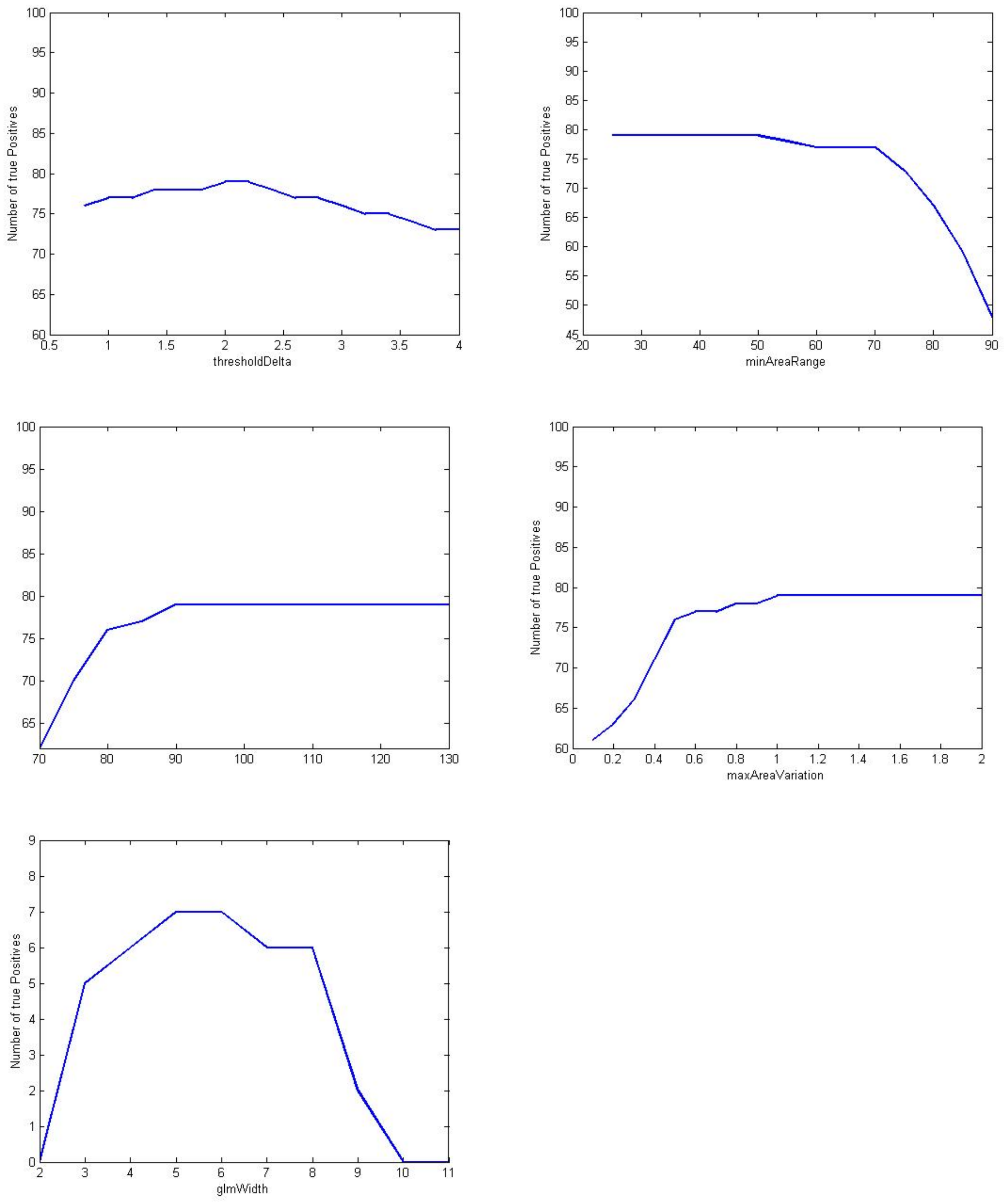


Figure 3.5: Number of cases in which our drain is identified as one of the candidate regions after varying each parameter variable.

3.4 Remaining Regions

As discussed in Chapter 2.6, when MSER is unable to find a suitable region within the required constraints, neither using the gray-scale version of the image, nor after performing dilation with a structural element, and the region in question is very dark, we try to use a very simplistic approach by applying a very low threshold of 39. After this, we perform despeckling. The value of 39 has also been reached after multiple tries, with 39 proving to yield the best results (out of 10 images in which the drain is covered by a shadow, we are able to correctly identify 7 of them), values between 36 and 41 giving fairly consistent accuracy. This approach is however extremely naive and susceptible to changes in brightness and we shouldn't expect much improvement after performing this last step. Furthermore, if one of our goals would be to minimize the number of false positives, it would be better not to apply this step at all. For a visualization of accuracy in this case, given different values for the aforementioned threshold, see Figure 3.6 below.

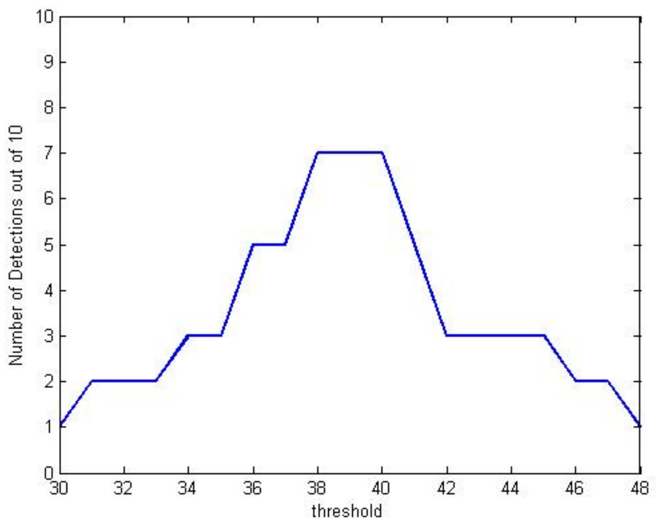


Figure 3.6: Performance of naive thresholding

3.5 Results

The detection was performed on all three test sets, while running the process 20 times so as to use different GPS estimates. We consider a drain correctly detected if it is within a circle with radius=10 pixels, which is only slightly bigger than the radius of the drain, thus our tolerance for error is very small. Below is the accuracy of detection for all three sets.

	Easy	Medium	Hard
Mean% detected ± Std. dev.	97.5 ± 1.4593	81.1764 ± 1.6098	46.5217 ± 0.9608

Figure 3.5: Accuracy of detection in all 3 sets given as a mean of the results obtained after running the detection process 20 times ± the standard deviation, for each run randomizing the GPS estimates of the drains and the road edges. The detection for the Easy set performs quite well, since none of the images in this set present any shadows, with both the drain and the edge being clearly distinguishable. For the Medium set, we have a few issues with applying the gray morphology in places where it is not needed to, or not applying this process since one or more regions respecting the constraints are found before exhausting the parameter iteration. For the Hard set, the most problematic type of image is the one in which the drain is either completely covered or immediately next to a shadow.

Chapter 4

Conclusions and Future Work

This project's goal was to analyze different methods of detecting road gully drains and put together a work-flow for achieving this, while carrying out an analysis on a 10000x10000 pixel image of a map containing 101 drains, each of which presenting different difficulties. While we achieved a reasonable detection percentage, a more in-depth analysis of the methods is necessary, with further methods being able to be implemented for an increase in accuracy.

Furthermore, despite us randomizing the available 'GPS' data, a more in-depth analysis of the approach, using more gully examples would be useful for statistical purposes. Some of the methods, such as applying gray-level morphology or detecting drains covered by shadows, were only tested on a small percentage of the available data, thus making the parameter optimization fairly naive.

A major drawback of the approach is the fact that it does not use any method of identifying when the gray-level morphology should be applied so as to fill in a shadow created by the pavement, instead applying this method when the MSER algorithm fails to detect any regions satisfying the required constraints. This in turn can lead to having a set of candidate regions that do not contain our drain, but only false positives. Similarly, it may lead to us applying the morphological dilation to images that do not require such a procedure. To counteract this, a method to detect when a long, dark line with an inclination and position similar to that of the GPS estimate is present in the image might prove useful. This could possibly be achieved by using a Canny edge detection algorithm or Hugh Transform [11], for which a Matlab implementation is available. If we would be able to accurately detect when to use a dilated image using the aforementioned gray-morphology, we could increase the percentage of detected drains within the Medium and Hard datasets, this issue proving to be one of the main causes for faulty detections.

Since the main reason for the inaccuracy in the Medium and Hard sets is the presence of shadows, we could also address this if we had multiple images of the drains taken in different times of the day. This way, we could possibly determine which version of the region containing the drain we are trying to detect is less shadowy and perform detection on that.

A further method which might prove useful, as it has already been used for drain detection [4] is analyzing the change in intensity of the area containing the identified MSER region. This could help us perform better filtering of the regions. For instance, if a region with a very low intensity in the center, getting progressively brighter towards the exterior is found next to a long, thin area of relatively high intensity, we could say with good certainty that it is a drain. An example of this is show in Figure 3.5 below.

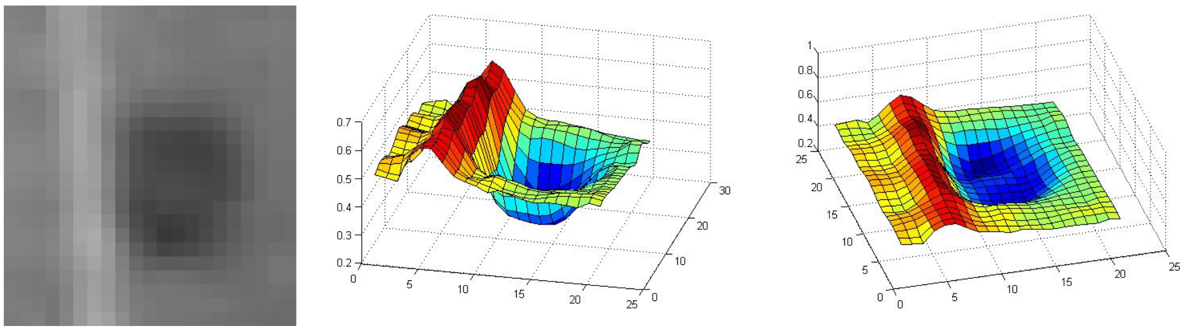


Figure 3.5: Manhole cover: intensity image (left), intensity plots viewed from different angles (center and right)

From a evaluation point of view, the approach we took to finding the optimal parameters might not lead to the best possible result. The optimal approach would have been to test the RANSAC and MSER algorithms using all possible combinations of the parameters and select the combination giving the best percentage of detection, while also having stable results if slightly varied. However, due to time constraints and the fact that the RANSAC algorithm has 3 parameters and the MSER one has 5, if we were to test 10 values for each parameter, it would amount to 10^3 and 10^5 runs for the RANSAC and MSER algorithms respectively. Knowing that both take an average of 10 minutes to run, the time it would take to analyze the two would be roughly 7 days for RANSAC and 694 days for MSER.

Detection of road gully drains can be a difficult task, especially since in order for the process to be used, the algorithm must prove to be very accurate, while having a low number of false positives. We have presented an approach for automatic detection of rectangular drains placed near the edge of the road, based on aerial imagery and GPS estimates, able to achieve very good detection in favorable conditions (97.5% for the 44 regions categorized as easy), with an overall average of 80.39%. Provided that more research is done on this topic and more data is available, it could prove a feasible solution, saving not only time, but also resources.

Bibliography

[1] M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Comm. of the ACM, Vol 24*, 381-395, 1981

[2] J. Matas, O. Chum, M. Urban, T. Pajdla. RobustWide Baseline Stereo from Maximally Stable Extremal Regions. *British Machine Vision Conference* 384-396, 2002

[3] J. Shunping, S. Yun, S. Zhongehao. Manhole Cover Detection Using Vehicle-Based Multi-Sensor Data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXIX-B3, 2012 XXII ISPRS Congress,* 25 August - 1 September 2012, Melbourne, Australia

[4] C. Drewniok, K. Rohr. Automatic Exterior Orientation of Aerial Images in Urban Environments. *Int. J. Comput. Vision*, 24:187-217, September 1997

[5] R. Timofte, L. Van Gool. Multi-view Manhole Detection, Recognition, and 3D Localisation. *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference,* 188:195 6-13 November 2011, Barcelona, Spain

[6] T. Ferreira, W. Rasband. ImageJ User Guide IJ 1.46r, *http://rsb.info.nih.gov/ij/docs/guide/user-guide.pdf*, 29.4:107 October 2012

[7] Y. Yu, J. Li, H. Guan, C. Wang, J. Yu. Automated Detection of Road Manhole and Sewer Well Covers From Mobile LiDAR Point Clouds. *IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, VOL. 11, NO. 9*, September 2014

[8] ImageJ Menu Documentation, *http://rsb.info.nih.gov/ij/docs/menus/image.html#type*, March 2015

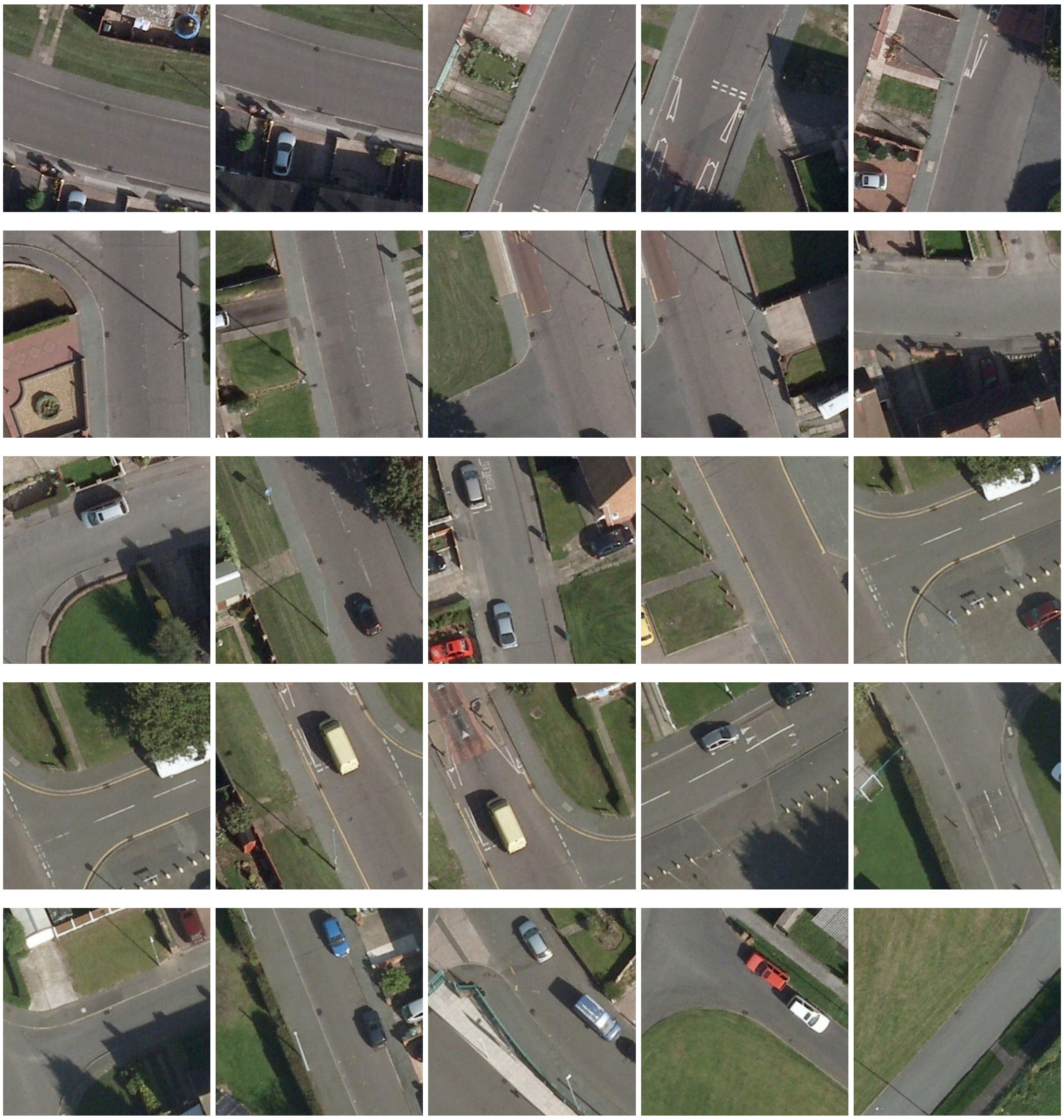
[9] Mathworks Documentation, *http://uk.mathworks.com/help/vision/ref/detectmserfeatures.html*, March 2015

[10] Mathworks Documentation, *http://uk.mathworks.com/help/images/ref/strel.html*, March 2015

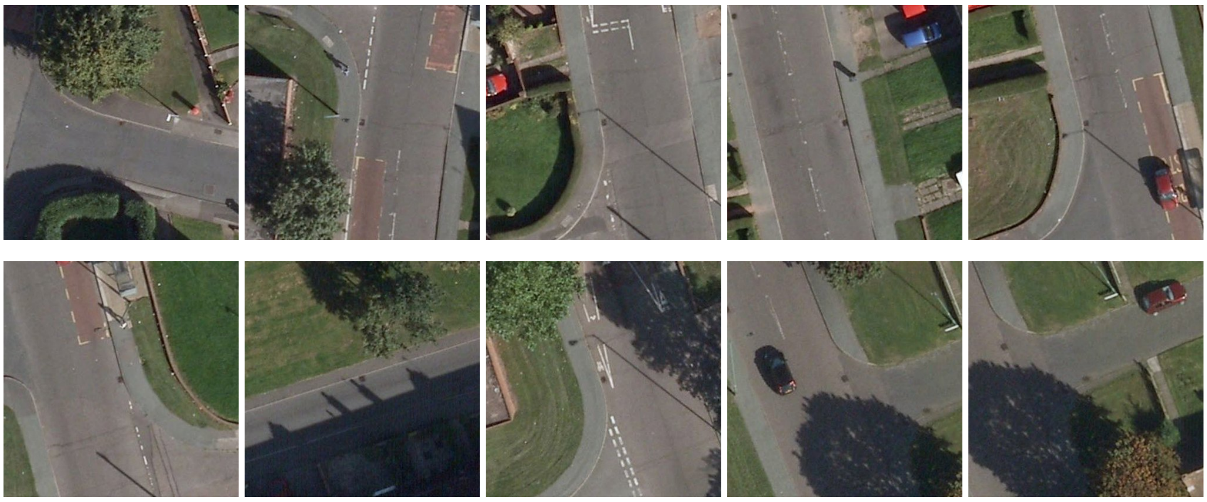
[11] Mathworks Documentation, *http://uk.mathworks.com/help/images/ref/hough.html*, March 2015

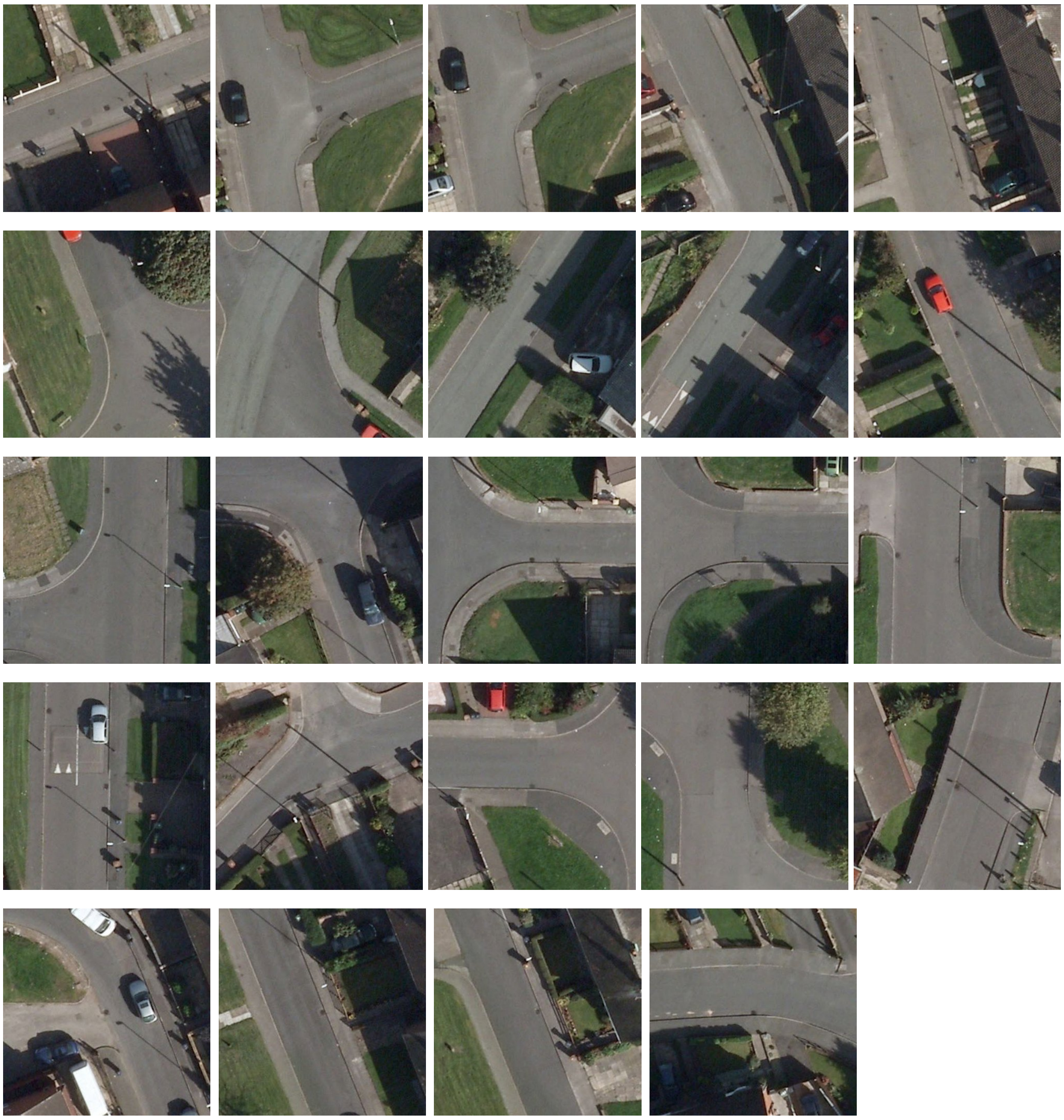
Index

Easy Set



Medium Set





Hard Set