University of Canterbury

A robust wire detector for a vine pruning robot

COSC460 : Honours Report

Josh McCulloch
10/6/2013

Abstract

An automated vine pruning robot is being developed to reduce the cost of labour in vineyards. This automated system requires an accurate model of the vine's structure, including the locations of support wires, in order for the robot to make good decisions about where and how to prune the plant. In this project we have developed a system for accurately and robustly detecting pixels belonging to wires in Bayer Images taken by the robot of the vine's canopy. Our system uses support vector machines for classifying wire and non-wire pixels, and a set of masks for optimally distributing training examples over an image. We have found an optimal subset of features for describing these examples and are able to achieve upwards of 90% precision with more than 20% recall. The system generates data ideal for wire fitting and use by the automated vine pruning robot. The techniques discussed could be generalised and used in other scenarios where selecting ideal example data from a large pool of potential examples, and finding optimal features to represent these examples is required.

Contents

Introduction

Vineyards require a significant amount of human labour to operate and pruning can consume 30-50% of the annual labour cost (1). In an attempt to reduce the burden of finding and training a workforce for this seasonal task, a team at the University of Canterbury is developing an automated vine pruning robot. The robot straddles the row of grapevines and will prune the plants using a computer controlled robotic arm. In order for the robot arm to be able to navigate the canopy of the grapevine and make good decisions on where to prune the vine, a comprehensive model containing the structure of the canopy needs to be created. This model needs to include the plant structure, post locations, and the wire positions. In this project we are specifically interested in accurately locating the wires in 2-Dimensional images of the canopy. Wire locations are found by first finding the pixels that represent wires in images captured by the robot. These wire pixels can then be used by the wire fitting system to find the wire locations. These wire locations are added to the canopy model for use by the rest of the automated vine pruning system.

The current automated vine pruning system contains a subsystem for locating the wire locations, but it uses a heuristic approach and can fail under some conditions. A system is required which can robustly find pixels which represent wires under varying and more difficult conditions. The wire detection system discussed focuses on finding the individual wire pixels for use by the existing wire fitting system.

Background and Related Work

Motivation

The goal of this project is to develop an effective robust system for extracting the wire locations in the canopy of the vineyard using images captured by a vine pruning robot. The wire locations need to be determined so an accurate model of the vine's canopy can be constructed. The model is used by the robot in order to determine where on the plant pruning should occur and how to navigate the canopy to perform the cuts. Finding the accurate locations of the wires is crucial to ensure that the robot does not mistakenly cut one of the wires or become entangled in them. The wires in the vineyard support the weight of the vines and can be under high tension; cutting them could damage the plants or the robot arm making the cuts. The arm becoming entangled in the wires will almost certainly cause damage to the robot, resulting in a significant loss. The robot is being designed to automatically prune vines which use the Vertical Shoot Positioning (VSP) training system containing between five and seven wires (2); all of which need to be accurately located.

Vine Pruning Robot

The vine pruning robot is under heavy development by the University of Canterbury and several industry partners. It is being developed to automate the pruning process in vineyards which is currently entirely performed by manual labour. Vineyard pruning is seasonal work which requires a significant level of skill, and a sizable workforce needs to be obtained and trained each year to perform the task. Managers of vineyards are regularly forced to compromise on the skill of the acquired labourers in order to obtain enough staff to perform pruning during the optimal time window; this can lead to a sub-par level of pruning resulting in reduced crop yield and quality. This goal of the Vine Pruning Robot is to create and autonomous system capable of pruning to a level comparable to that of an expert veteran pruner, and could prune a significantly larger number of vines per day than its human counterpart.

Related Research

Detecting High Tension Wires

Methods for detecting high tension suspended power-lines have been implemented in collision avoidance systems on board aircraft. The Passive Obstacle Detection System (PODS) (3) for Wire Detection developed by Boeing is able to detect suspended wires at ranges greater than 4.5 kilometres using both visible and long wave infrared light. PODS uses a multistage pipeline to first remove noise from the image before slowly building up a model of wires in the image from primitive components. PODS first applies a ring median filter to remove any large scale structures from the image. The median filter has a radius of 5 pixels; wires are assumed to have a diameter of less than this. The wires generally have a lower contrast than other elements in the image and so a SUSAN filter is applied to create an image containing just the high contrast elements. This high contrast image can then be subtracted leaving only the lower contrast elements. The system then calculates the gradient phase operator using 2 convolution kernels; one for the horizontal direction and the other for the vertical. This results in an image containing small scale gradient changes across the image. The gradient along the top edge of the wire should be opposed to that of the lower edge (see Figure 1). By looking for pixels with neighbours with diametrically opposed gradient phases, potential wire segments can be found. 16 successive linear filters are then applied to the image to find the probability that a wire segment belongs to a wire based on its neighbouring pixels. Varying

thresholds are applied with each filter leaving only pixels with a strong likelihood of being part of a wire and rejecting segments which are disconnected.

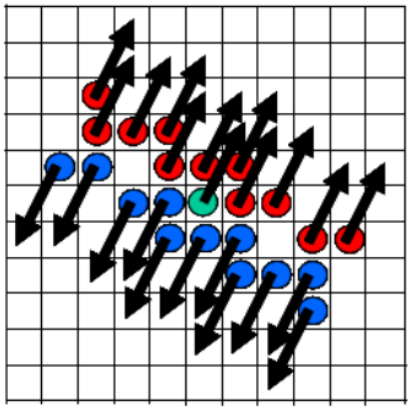


Figure 1: Gradient phases of the upper and lower edges of a wire

The images captured by the vine pruning robot include the plant's tendrils. When viewed in isolation these tendrils have many similarities to the wires in the image. PODS does not need to deal with such a similar objects to those that it is attempting to be classify in its images. Our system needs to be able to robustly classify the wires, while rejecting the wire-like tendrils.

Detecting Wires in the Vineyard

The previous system implemented on the vine pruning robot uses a heuristic based approach for detecting wires. This system, along with many other computer vision based systems on the robot, works with background subtracted images. These background subtracted images work by calculating the probability that a given pixel is part of the foreground using Bayes formula. The system looks at each pixel individually and compares it to an image of the background. By looking for differences between these images the probability that a given pixel is part of the foreground can be calculated. The current wire detection system assumes that wires will be between one and six pixels in thickness and approximately horizontal. It works by looking at the pixels above and below the pixel of interest, if they are part of the background while the pixel of interest is part of the foreground, then it is likely that the pixel is part of a wire. Long thin canes can meet this criterion, so the system also looks at the colour of the pixel; wires are generally grey while vines are browner. The system then sweeps across the image looking for pixels that are co-linear and forms wire segments from them. The system then connects these wire segments to identify the wires. The final sweep and co-linear search removes almost all false positives and the main concern with the system is its lack of sensitivity under some conditions. Sometimes the wires are not well represented in the background subtracted images or even non-existent.

Road Lane Detection

Detecting lane markings on roads bears some similarity to wire detection. Lipski et al. have developed a fast and robust method for detecting lane markings using multiple cameras and parallel processing on Graphical Processing Units (GPU) (4). Their method takes four images of different view perspectives and creates a large mosaic image for processing. The lane model is reconstructed from a set of lane segments provided by the feature detection system. Feature detection is processed in parallel on the GPU to reduce classification time. The feature detection algorithm works with eight by eight pixel sub regions from the large mosaic image. Each of these sub-regions undergoes a series

of tests in order to determine the likelihood that the given sub-region contains a lane segment. These tests include checking the distribution of colours and the shape of the pixels inside the segment. The lane segments created by this process can then be fitted into the lane model.

The assumptions made by the lane detection system are somewhat analogous to those we can make when detecting wires. Lane detection needs to allow for obstructions hiding lanes from the cameras; our system will also need to allow for objects obscuring the wires from view. The lane detection system needs to allow for varying lighting conditions. The vine pruning robot provides its own light source so this is not an issue. However we do need to allow for varying colours of wire and plant structure, which if unaccounted for could be detrimental to the robustness of the system.

Classification

Support Vector Machines

Support Vector Machines (SVMs) have been used by the vine pruning robot for recovering laser lines projected into the vines' canopy. This structured light is used by the system for creating a 3-dimensional model of the canopy structure. Botterill et al. achieved a 99% precision with 90% recall when detecting the structured light in the canopy (5). Achieving similar results would greatly benefit the wire detection system. However detecting a laser is generally simpler for a classifier than detecting wires due to the high contrast and difference in hue between the laser and the rest of the canopy. SVMs generally achieve similar results to other machine learning techniques such as Neural Networks and Random Forests (6) (7). SVMs work by attempting to find a splitting plane which divides the positive and negative examples (8). This plane can be linear or have a more complex shape to better match the examples. The training vectors can be plotted in n-dimensional space where n is the number of features that a vector has. A hyper-plane can then be found which splits the examples with the maximum separation. In instances where an ideal splitting plane cannot be found the vectors can be mapped into a higher dimensional space with the use of a kernel function; these higher dimensional spaces allow for more complex splitting planes to be found. The Gaussian Radial Basis Function (RBF) kernel can approximate a large number of smooth splitting surfaces allowing it to perform well in most cases. The complexity of the splitting plane found by the RBF kernel can be adjusted, a complex splitting plane runs the risk of over fitting, while less complex splitting plane may over simplify and not correctly describe the data.

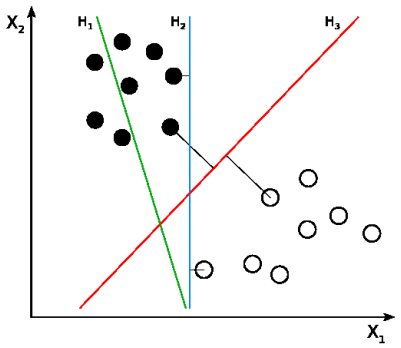


Figure 2: Example of 3 splitting planes. Both H_2 and H_3 successfully split the data set while H_1 does not. [wikipedia.org]

Neural Networks

In earlier research, we used Neural Networks (NN) (9) for identifying wire locations in images of the grapevines (10). We used four NNs; one for each colour channel of the Bayer Image. The NNs were

trained on randomly selected positive and negative examples from one image of the vine. The output from the NNs was then passed through a threshold function which looked at neighbouring pixels. If there were more neighbouring wire pixels than the pre-set threshold, the pixel was classified as a wire. This threshold function greatly improved the precision by removing a large number of false positive. This system ultimately achieved a precision of 94% which would generally be good enough to accurately fit wire segments to. Half of the remaining false positives were caused by a laser line that was projected into the scene and could potentially be removed with further training of the networks. The major issue with NNs is the long time required for them to classify an image; taking 27 seconds for a 960 by 1280 pixel image. This could be reduced by intelligently testing pixels, for example based on whether neighbouring pixels had already been classified as wires. However NNs still require a large amount of computation to classify an entire image.

Heuristic

Applying heuristics to classification problems is generally computationally efficient and fast (11). There are many heuristics that could be used to identify potential wire segments (12), such as testing the colour, as wires tend to be a shade of grey, or shape, as wires are thin structures and usually close to horizontal. The existing system uses a scale invariant, shape based heuristic which is looking for thin horizontal structures against the background. The wires are generally between one and five pixels in diameter and can be extracted with a relatively high level of accuracy. False positives can be further removed by examining the colour of classified pixels, as wires are generally greyer than other structures in the image. While the heuristic based approach is computationally light compared to using machine learning techniques, it does have some pitfalls. The heuristics need to be programed into the system; if the environment is prone to change this can require that the system be regularly reprogrammed. A classifier based on machine learning techniques could be retrained using some current example images rather than requiring reprogramming. Heuristics can also be used alongside a dynamic system utilising machine learning techniques. Heuristics could be used to pre-process an image and identify candidate pixels that would then be checked by the classifier. This method of boosting could improve the computational efficiency of the system and remove objects that are easily extracted such as the blue backdrop; removing the requirement that the machine learning classifier discovers this relationship.

Input Data

Bayer Images

Raw image data captured by the vine pruning robot is in the form of a Bayer Image. In most cameras today each pixel is only capable of capturing one colour; be it red, green, or blue. The colours of the pixels are generally laid out in the Bayer Pattern, developed by Bruce Bayer (13). In the Bayer Pattern every odd numbered row will alternate between green and blue pixels, while every even row alternates between red and green pixels. This pattern contains as many green pixels as blue and red pixels combined; this is to mimic the physiology of the human eye. In this raw data captured from the camera, each pixel contains a single value; the colour this value represents depends on the pixel's absolute position in the image (see Figure 3 and Figure 4).

Figure 3: Example of a Bayer Image. (Colours are used to show the colour represented by each pixel) [wikipedia.org]

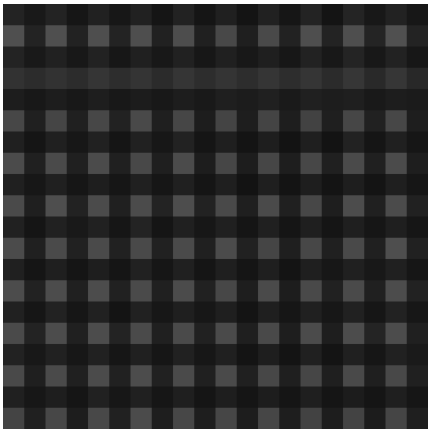


Figure 4: Example of a Bayer Image used by the system. A wire can be seen across the top of the image

Colour Images

The colour images used on the vine pruning robot are generated from the raw image data (see Bayer Images). The process of reconstructing the colour image from the raw image data is called demosaicing (14). The demosaicing process finds the values for the two colours of a given pixel that it doesn't contain, for example the red and blue values for a green pixel. These values can be found by bilinearly interpolating the colours of neighbouring pixels. For example, for a pixel that only contains a green value, its red value can be found by considering the two red pixels either above and below or to the left and right of the green pixel; the blue value would then be found using this same method. The process of reconstructing a full colour image by demosaicing a Bayer Image results in a lower effective resolution (see Figure 5) than what would be achieved by using three separate image sensors; one for each colour.

Figure 5: Difference in effective resolution caused by demosaicing [wikipedia.org]

Background Subtracted Images

The background subtracted images contain the probabilities that a given pixel belongs in the foreground. Hence they in theory have had the background subtracted from them. These background subtracted images are generated by the existing system used on the vine pruning robot (15) (16). These images (see Figure 6) are used by various subsystems on the robot such as the cane reconstruction, and wire extraction systems. These images are made by comparing the current frame with an original frame which did not contain the canopy (vines, posts, wires, etc.). The probability that a given pixel is part of the foreground can then be calculated by comparing its value in the current frame with that in the original frame. The background subtracted image does not contain any colour information; each pixel contains a single value representing the probability that is in the foreground of the image.

Figure 6: Background Subtracted Probability Image

Red Green Blue Colour Space

The Red Green Blue (RGB) colour space represents colours using the three colourants red, green, and blue. Three values are used to represent a colour in the RGB colour space; one value for each colour intensity. The RGB colour space is a convenient representation for colour information; this is due to the design of image source. The camera's sensor uses three different sets of sensors, as discussed in the Bayer Images section and is sensitive to one of the three colours Red, Green, and Blue. This mapping from sensor data to RGB colour space does require the lossy demosaicing process discussed in the Colour Images section. In Figure 7 we can see an example of the 3 individual colour channels of an image used by the vine pruning robot. The wires generally have a grey colour which results in them having somewhat equal intensity values in each of the three colour channels.

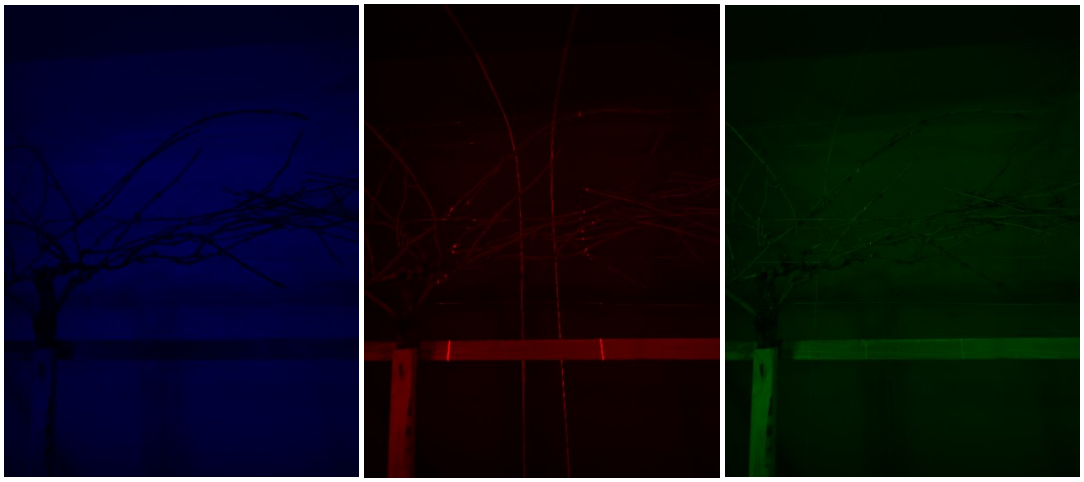


Figure 7: Examples of the Red, Green, and Blue colour channels

Hue Saturation Value Colour Space

Hue Saturation Value (HSV) is a cylindrical coordinate representation of the points in the RGB colour model. The Hue value is used to select a colour by referencing a position around a colour wheel. The saturation value represents how saturated the selected hue is. The "Value" value is used to represent the brightness of the selected colour; where a small value would result in black and a large value would be the original colour. The HSV colour space is of particular interest because it separates the colour and intensity values; this was not the case in the RGB colour space. In Figure 8 we can see each of the three HSV values along with the original full colour image. We can see the Hue image allows us to easily isolate the plant structure from the background, but does not contain a strong

representation of the wires; however in the Saturation image the wires are much more prominent. By combining the information in each of these channels we could envisage a system which could progressively remove parts of the image, first the plant structure, then the background, until only the wires remained. The difficulty in a system like this is determining the threshold value for describing each of the objects in the image; for example in the saturation image the wires appear visible, but a change in lighting or wire reflectivity could change how the wires are represented or even completely remove them from the image.

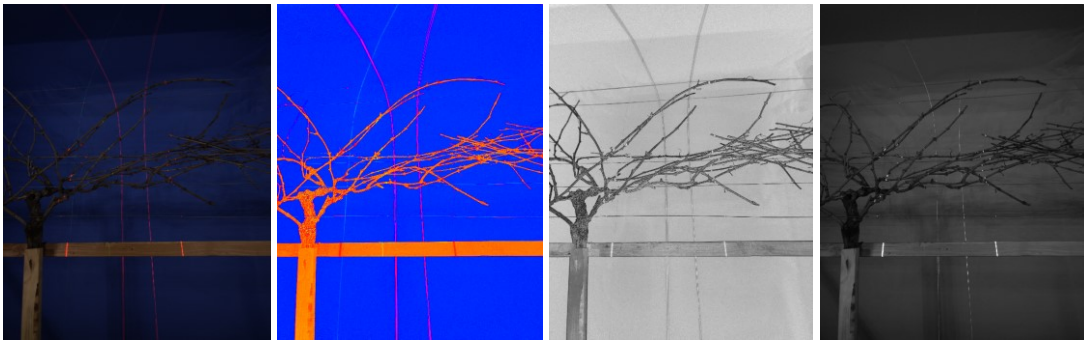


Figure 8: Original RGB Image split into separate Hue, Saturation, and Value channels

Fitting Wires

The vine pruning robot already has a robust method for fitting wires to the classified data based on the least squares algorithm (17). We however briefly cover several methods for fitting the wires for completeness.

Linear Regression with Least Squares

Linear Regression using Least Squares is a fast and computational efficient method for calculating a straight line through a set of data points. The method first takes the mean of the data set which the final model is expected to intercept. This then constrains the problem, as only the slope of the line now needs to be found. The slope can be found by measuring the sum of the squared distances the X parameter of each data point varies from the means, and then repeated for the Y parameter. This method of fitting a line to the data points requires that only one wire exists in the data set. Linear regression is also affected by outliers.

RANSAC

RANSAC or Random Sample Consensus is an iterative method of fitting a mathematical model to a data set which contains outliers. RANSAC operates by first selecting a random subset of examples from the data set. A model is then fitted to these examples. Every example that is not part of the initial random subset is then tested against the model, if the example fits the model it is added to the consensus set. This process is repeated, each time with a different initial subset of examples. For each of the iterations, the size of the consensus set produced by the model is recorded. The model with the largest consensus set is considered to best represent the data.

Hough Transform

The Hough Transform is a technique used for feature extraction based on a voting procedure. For each data point presented to the algorithm, all possible model parameter sets which fit are given a vote. After all of the data points in the data set have been processed, the parameter set with the greatest number of votes is considered to best fit the data set. Figure 9 shows a set of potential wire

segments that need to have wires fitted to them. Note: there are a significant number of outliers in this data set. In Figure 10 we can see each of the data points plotted in parameter space. For each data point every possible line that intercepts is considered. The parameters of these lines are plotted in parameter space; the angle of the line is mapped onto the x-axis and the distance that line is from the centre of the images is mapped onto the y-axis. With this method of mapping each data point produces a series of votes which appear sinusoidal. Parameter sets which more commonly match the data, gain more votes and create a local maximums. These local maximum can then be projected as lines back into the data set space; see the yellow lines in Figure 9. In this case the outliers in the data set have not affected the fitting process because there were enough inliers to disregard them. The Hough Transform performs well for detecting multiple lines from the same data set. By taking the number of wires expected in the data set which is known, we look for that number of local maxima in the parameter space.

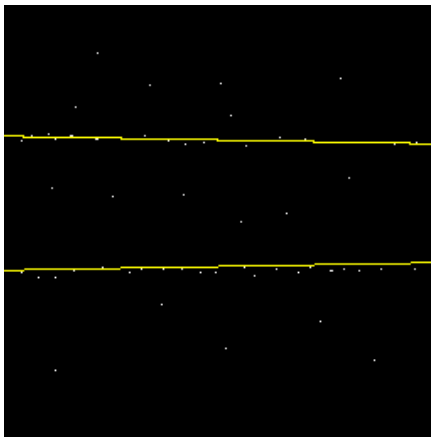


Figure 9: Data space with two lines re-projected from parameter space

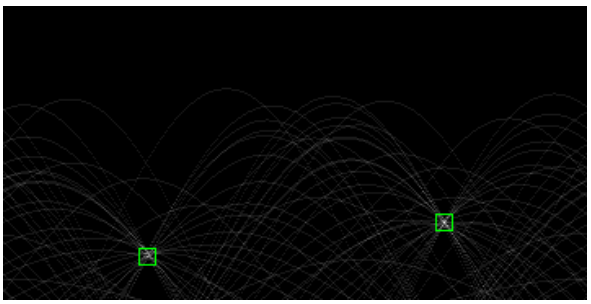


Figure 10: Parameter Space with Theta mapped onto the X axis and Radius onto the Y axis

System Design

System Input

Bayer Images

The system feeds Bayer Images into the classifier. Bayer Images offer several advantages over traditional colour Images. The colour images (see Colour Images) provided by the system contain three values for each pixel; one for each colour. While the Bayer Images only contain one value (see Bayer Images), it contains at least as much information as a colour image, as the colour image is created from it using a demosaicing algorithm (see Figure 11). We opted to base our system around these images because the Bayer Image offers raw data untouched by post-processing such as demosaicing. Bayer Images require fewer values to be fed into the classifier used to identify wire pixels as each pixel only has one value. By reducing the amount of data fed to the classifier we can reduce the risk of over fitting. The Bayer Image does however require four classifiers to be created; one for each of the four colour locations (see Figure 13).

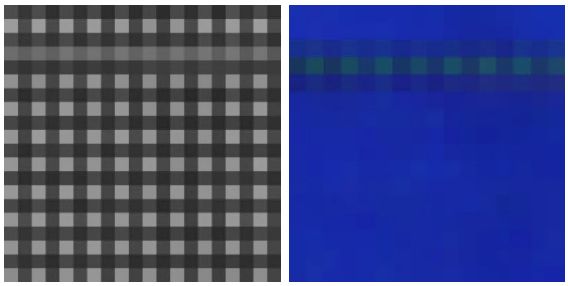


Figure 11: Comparison between Bayer and colour images containing the same segment of wire

Marked Images

Each Bayer image provided to the system was accompanied by a corresponding marked image. The marked image was created by humans and was considered to be the ground truth. The marked images contained several colour masks which isolate different regions to then be used during the example creation phase. The wires were marked in the image with white lines with a width of 1 pixel. This one pixel width line did not mask out the entire wire because the wire generally has a diameter of 3-5 pixels. To completely mask out the wire, a border of 2 red pixels was applied to the white line. When the system is looking for positive examples it references this marked image looking for white pixels. Any pixel that is not white or red in the marked image is a candidate negative pixel.

To control the distribution of negative examples we added two additional masks to the image. The first mask was magenta and was used to mask out posts and other support structures (see Figure 12). The second mask was blue and was applied to the backdrop in the frames provided by the robot. By adding these two additional masks we can choose the distribution of negative examples that fall on the backdrop, posts, and the vine. Being able to adjust these distributions is important because the backdrop alone takes up more than 80% of the image by area while the vine structure is the most complex; requiring significantly more training examples to identify. If we were unable to adjust these distributions we would have a large number of examples available to learn the simple structure of the background and a small remaining set of examples to learn the complex structure of the vine. In our default distribution, 10% negative examples fell on the backdrop, 10% on the posts and support structure, and the remaining 80% fell on the vine.

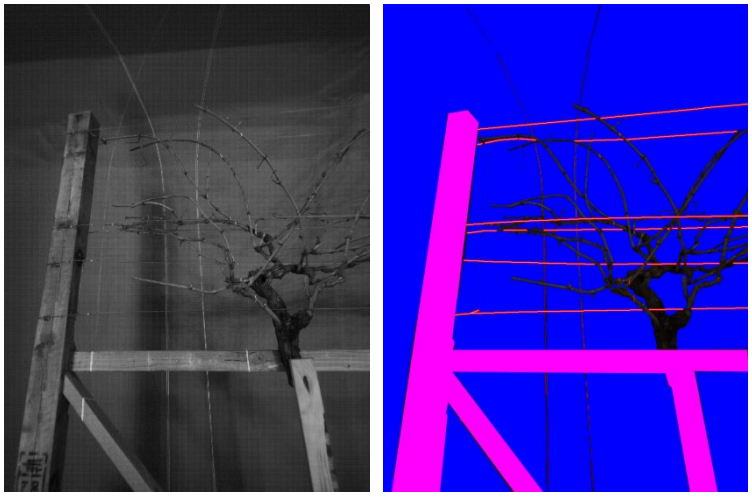


Figure 12: Bayer and corresponding marked image

Support Vector Machines

The system uses four SVMs, each bound to a separate colour channel in the Bayer Image. A separate SVM is required for each channel because the vector creation process is based on the layout of the neighbouring pixels of the pixel being classified, for example depending on what channel is being classified the pixel to the left of the pixel of interest will represent a different colour. The solution we used to deal with the data, being location dependent, was to use separate SVMs for each channel. This required the SVMs to be individually trained and tested as well as each having their own set of features they used during the creation of vectors. Figure 13 shows the relative position of the four SVMs.

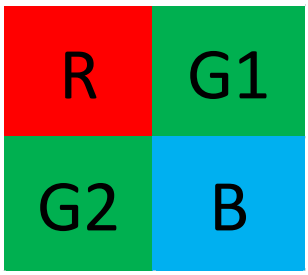


Figure 13: Relative layout of the four SVMs

Creating Vectors

Vectors

A vector is an n-dimensional set of values representing some object. Each vector is constructed by extracting a set of features which describe the object in question. Many algorithms in machine learning and pattern recognition require information to be provided in this form. Each feature inside the vector may represent a single attribute about the object such as a pixel's brightness, or a combination of attributes such as the mean brightness of all the pixels in a given location. A vector can have a known class, such as wire or non-wire, or its class can be unknown in which case the vector is considered un-labelled. When a vector has a known class it can be used for training the classifier to recognise patterns and associations. Once a classifier has been trained it can then be feed un-labelled data to classify. Because the training data used by the classifier needs to be labelled

we need to find a way to identify the class of the vectors. In our work the training data was extracted from images for which the wire locations had been identified by humans.

Features

In order for the classifier to identify whether a given pixel is part of a wire, we needed to develop a system to create vectors that are able to describe the pixels. In the Bayer Image each pixel contains a single byte on information which may represent one of three colours depending on its position in the image. If we used this single byte to create the corresponding vector, the vector would only contain one element and not enough information to identify the class of the pixel. In order for a pixel to be classified we need to look at the attributes of neighbouring pixels. In this project we consider all pixels in an eight by eight grid centred on the pixel of interest (see Figure 14) when creating the describing vector. The 64 pixels inside this region of interest could be each treated as a single feature, or combined to create a lower dimensional feature.

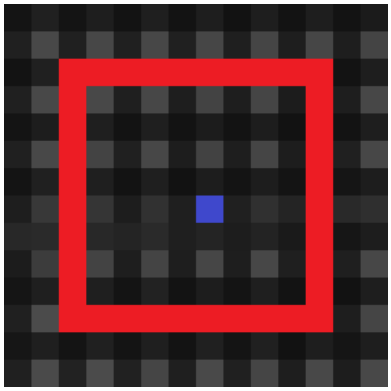


Figure 14: Neighbouring pixels inside the red region are used for creating the vector for the blue pixel of interest.

Creating a new feature for each neighbouring pixel passes more information to the classifier than a smaller set of features built by combining pixels, however there are reasons why combining features is preferable. Firstly by building and combining features we reduce the chances of the classifier over fitting. Building and combining features abstracts away smaller details and can leave bigger more important structures for the classifier to learn from. The second reason for combining features is that it can reduce the total number of features in each vector. The more features that are used during classification, the more computationally expensive each classification is. Also by reducing the number of features in each vector the classifier can require less training examples before it reaches an acceptable level of performance.

Trees

In order to generate features we needed to develop a system which could describe how and which individual pixels would be combined. We wanted a system that would be robust and could be tuned using meta-learning techniques such as genetic algorithms or an exhaustive search. One such abstract data structure that was investigated was the tree. Trees could pull from a library of base features such as rows, columns, individual pixels, and regions of pixels, and describe how to combine them using arithmetic operators such as addition and subtraction (see Equation 1), or logical operators such as, ‘and’, ‘not’, and ‘or’.

Equation 1: Example of feature created from a tree representation

$$feature = \big(allPixels - \ column(3)\big) - row(2)$$

Trees have the advantage of having large amounts of research exploring their application in describing solutions, such as linear regression and decision trees. Trees can have many techniques applied to optimising them such as linear regression and pruning. The major downside to using a tree representation for describing features is that they can be slow to traverse and with the large amount of searching and classification required in this project, complex trees would make it computationally infeasible to obtain large datasets for analysis in a tangible amount of time.

Masks

Masks are conceptually less complex than tree structures and ultimately cannot be described as complex relationships. The masks used in this experiment were eight by eight 2-dimenisonal arrays that masked out individual pixels. The mean value of the pixels that remained after the masking process would then be used as a feature. While masks may not be able to describe as complex relationships as trees can, they do still provide a large possible feature space, and do have the massive advantage of computational efficiency. These masks can easily be represented as a 1-dimensional bit string allowing for genetic algorithms to be easily applied should we wish to perform a more extensive features search.

Feature Library

To reduce the search space of possible features to a tractable size we opted to create a feature library and search for an optimal subset of features from within it. This feature library contained 96 features from the 64 neighbouring pixels. 64 of the 96 features were direct representations of the 64 individual pixels; the remaining 32 features were created by taking the mean value over the pixels in each row and column. Because each row and column in the Bayer Images contains two colour channels, two features were created for each one. In this implementation one feature might contain the mean blue value for a column, while the other would contain the mean green; preventing the colour channels from being combined. Early experimentation showed that merging the colour channels resulted in a significant decrease in the performance of the classifier.

Vectorisers

The vectorisers are an abstraction used to convert the 64 neighbouring pixels into a vector which could then be used during the training or classification processes. During the development phase we created multiple vectorisers. These vectorisers allowed us to test the effectiveness of different methods of feature creation such as taking the mean value of a set of pixels. The final vectoriser was implemented using the previously discussed masks. When provided with raw data from the Bayer Image and a set of masks, the vectoriser would iteratively apply each mask and compute the mean pixel value. Each mean pixel value would be added to the final vector used as a feature to describe the pixel currently being classified.

Finding Examples

The data creation pipeline (DCP) creates examples used for training and testing the classifier. Each example produced by the DCP contains two parts, the first is the vector which contains the features describing the example, and the second is the class of the example; whether it is positive or negative. During the development of the system we created a revised version of the DCP; however

both versions maintained many similarities. Both versions used a previously marked image (see Figure 12) to identify the different regions of the image, and a Bayer Image to extract the raw data from. Both versions also used a vectoriser to convert the 64 pixels into a corresponding vector and could shuffle the data if it was required; such as with N-fold cross validation techniques.

Two subroutines were used by both versions of the DCP for finding a given number of wire examples and non-wire examples. These subroutines randomly find the locations of wire and non-wire examples by referencing the provided marked image. However the way the two DCP implementations used these subroutines differed. The initial implementation was provided with two numbers; the quantity of positive and negative examples to be found by the subroutines (see Figure 15). The revised implementation of the DCP removed these two numbers from the inputs and replaced them with two sets of examples; the positive and negative (see Figure 16). These two subroutines were removed from the DCP to allow for more control of the systems behaviour. The initial implementation would randomly find new example locations each time it was run. This resulted in additional variance, for example when trying to measure the effect of making changes to the vectoriser. By moving the subroutines out of the DCP, we are able to use the same example locations but use different vectorisers and thereby different vectors describing the same examples.

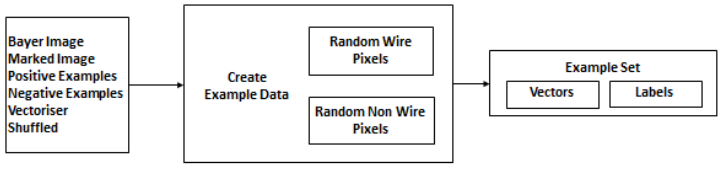


Figure 15: Initial Data Creation Pipeline

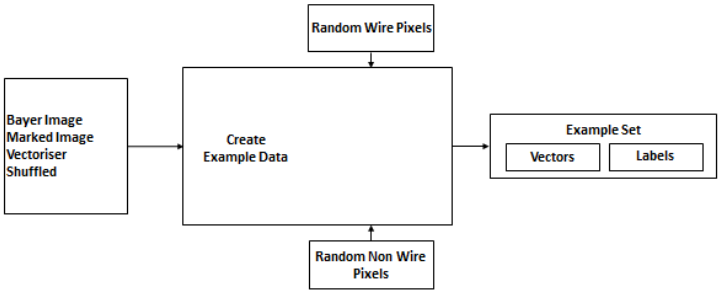


Figure 16: Revised Data Creation Pipeline

Measuring Performance

Marked Images

In order to measure the performance of the classifier we need to first determine what is and is not a correct classification. When we trained the system we only used positive examples that fell on the white pixels, however when it comes to measuring the performance of the classifier, any pixel that falls on the red border (see Figure 17) of the wire will also be counted as a correct classification. Using this method we can avoid the issues caused by the wire not being a constant width across a given frame and between multiple frames.

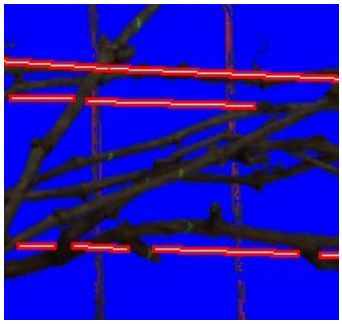


Figure 17: Marked wires with boundary guard area

Entire Image Classification

Initially we measured the performance of the system by classifying an entire image. This approach seemed logical because it is the task that the classifier would be expected to perform once it was deployed. However this method of analysis did have two significant drawbacks. Firstly the time taken to classify an entire image was about 4 seconds which made it computationally infeasible to run large experiments, for example when performing feature searches which have large search spaces. The second issue is that this method had a lot of variability between classification attempts making the process of fine tuning parameters difficult.

10-Fold Cross Validation

In order to run larger more accurate experiments it became apparent that we needed a new method to gauge the performance of the classifier. We opted for a K-fold cross-validation technique, specifically 10-fold which is a common approach to measuring the performance of classifiers. In K-fold cross-validation the samples are shuffled and split into k partitions of equal size. One of these k partitions is then used for testing the classifier while the remaining k-1 partitions are used to train the classifiers performance. This process is repeated k times with each fold being used once for validation. The samples are shuffled to ensure that each fold has approximately the same distribution of positive and negative examples.

We found that 10-fold greatly reduced the amount of variability between independent experiments when compared with the entire image classification method of measuring performance. By only using 10,000 examples in the 10-fold cross-validation, instead of the 1.2 million in the classification of the entire image, we dramatically reduced the running time required.

Performance Measures

We can directly measure the number of true positive, true negative, false positive and false negative classifications made by the classifier. From these four values we can then create more useful performance measures such as precision and recall. Initially we used the precision of the classifier to gauge its performance. However as we tuned the classifier in an effort to push the precision higher, we saw a dramatic drop-off in the overall recall of the classifier. If the classifier is achieving 100% precision but only classifying a few 10s of pixels as wires, we will be unable to fit wires to the points and the data will be effectively useless.

$$precision = \frac{tp}{tn + fp}$$

$$recall = \frac{tp}{tn + fn}$$

In order to maintain an acceptable recall level while driving up the precision, a performance measure that accounted for both the precision and recall of the classifier was required. The F-score is a commonly used measure of accuracy that meets these requirements. The F-Score is essentially a weighted average between precision and recall as defined in Equation 2. From the F-Score distribution in Table 1 we can see that the F-Score is low when either the precision or recall is low; larger F-Scores are only achieved when both the precision and recall values are large.

Equation 2: F-Score definition

$$fScore = 2 * \frac{precison * recall}{precison + recall}$$

Table 1: Resultant F-Score values for various precision and recall values

		Precision										
		0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Recall	**0**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.1	0.00	0.10	0.13	0.15	0.16	0.17	0.17	0.18	0.18	0.18	0.18
	0.2	0.00	0.13	0.20	0.24	0.27	0.29	0.30	0.31	0.32	0.33	0.33
	0.3	0.00	0.15	0.24	0.30	0.34	0.38	0.40	0.42	0.44	0.45	0.46
	0.4	0.00	0.16	0.27	0.34	0.40	0.44	0.48	0.51	0.53	0.55	0.57
	0.5	0.00	0.17	0.29	0.38	0.44	0.50	0.55	0.58	0.62	0.64	0.67
	0.6	0.00	0.17	0.30	0.40	0.48	0.55	0.60	0.65	0.69	0.72	0.75
	0.7	0.00	0.18	0.31	0.42	0.51	0.58	0.65	0.70	0.75	0.79	0.82
	0.8	0.00	0.18	0.32	0.44	0.53	0.62	0.69	0.75	0.80	0.85	0.89
	0.9	0.00	0.18	0.33	0.45	0.55	0.64	0.72	0.79	0.85	0.90	0.95
	1	0.00	0.18	0.33	0.46	0.57	0.67	0.75	0.82	0.89	0.95	1.00

Finding the ideal feature set

Finding the ideal set of features to be used for vector creation is crucial to optimising both the performance and classification time required by the classifier (18). Each additional feature used by the classifier increases the amount of time required to classify the example. Having redundant additional features also runs the risk of the classifier over fitting the training data. An example often used to describe this is the diagnoses classifier. *If a classifier is fed a set of parameters about a patient in order to diagnose the patient's illness, we wouldn't expect the patient's ID number to be any help in the diagnoses process, however if there is not enough training examples then the classifier may falsely identify a pattern in the patient ID numbers.* By reducing the number of features fed to the classifier we can reduce any redundant data being fed to the classifier (19).

We used two methods for finding the ideal feature sets for each support vector machine. Even with a feature library of only 96 features, there is a computationally infeasible number of ways of selecting a subset of features. In order to select a near optimal subset of features we used two greedy algorithms, *Add one in* and *Leave one out*.

Add one in method

The *Add one in* algorithm is a simple method of finding a small set of good features. Given the current set of features, individually test each of the remaining features with the current feature set. Select the feature that performed best with the current feature set and add it to the current feature set. This process is continued until the desired number of features has been selected, or the performance of the classifier has stopped increasing.

There is one significant draw back to the *add one in* algorithm and that is because no matter which features you select, the classifier cannot make any correct positive classifications when it has very few features. When the classifier is not making any positive classifications there is no way of distinguishing the difference in performance of various subsets of features. To allow the classifier to make some positive classifications we seed the algorithm with a small set of features which have performed well during the development phase.

Leave one out

The *leave one out* method is essentially the reverse of the *Add one in* method (20). It works by seeding the current feature set with all features. Then individually, each feature is tested to see what effect removing it has on the classifier's performance. The feature which has the least negative impact or most positive impact on the classifier's performance is then removed from the current feature set and the process is repeated. This approach has the advantage of not requiring a subset of features to first be selected by another means, removing the bias which is present in the *Add one in* method. However when a large number of features are present in the current feature set, removing one feature has a very small effect of the performance of the classifier. Therefore it is important to have an accurate way of measuring this performance. The entire image classification method did not achieve the required level of accuracy and it was not until we began using the 10-fold cross-validation technique that the *leave one out* method became feasible.

Finding the optimal SVM Hyper-parameters

The classifier used for this project was a support vector machine with a Radial Bases Function (RBF) kernel. The RBF kernel is used to map the features into a higher dimensional space where a splitting plane can be found. The RBF kernel has two hyper-parameters, C and Gamma, which need to be correctly selected in order to gain the optimal performance from the classifier (21). The C hyper-parameter is used to define the relationship between misclassifying training examples and the simplicity of the splitting plane. A high C value will result in a complex splitting plane and more of the training examples being correctly classified, while also risking over-fitting the classifier to the data. A low C value will result in a simple splitting plane, while potentially misclassifying some of the training data; this simple splitting plane may be too simplistic and not correctly describe the larger scale relationships in the data. The Gamma value used by the RBF kernel defines the region of influence of each training example. Small values of Gamma result in a small region of influence and large value in a large region of influence.

We used a 2-dimensional grid-search to find the optimal C and Gamma values for the SVMs (22). We searched for a C value between 2^{-5} and 2^{15}, and for a Gamma value between 2^{-10} and 2^{10}. In a grid search the columns represent one variable and the rows represent the other. In this experiment the columns represented the Gamma values from 2^{-10} to 2^{10} and the rows represented the C values from 2^{-5} to 2^{15}. The search algorithm then iterates of the grid and stores the classifiers performance for

the given set of parameters in their corresponding cell in the grid. A gradient descent method for finding the optimal set of values may be more computationally efficient; however a gradient descent algorithm can become trapped in local minima and are not easily parallelised. Our grid search algorithm recorded precision, recall, and F-Score values of the classifier. Alternatively the grid search can be performed using the empirical classification error as a performance measure.

Results

Table 2: Precision values from Hyper-Parameters Grid Search

C Exponent \ Gamma Exponent	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
-5	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
-4	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.50	.48	.25
-3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.50	.92	.91	.92	.96
-2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.99	.90	.89	.88	.88	.90
-1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.97	.89	.88	.87	.86	.86	.88
0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.25	.93	.88	.87	.86	.85	.85	.86	.88
1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.25	.93	.89	.87	.86	.85	.85	.85	.86	.88
2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.67	.92	.88	.86	.85	.85	.84	.85	.85	.86	.89
3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.67	.92	.88	.85	.85	.84	.84	.84	.84	.85	.87	.89
4	.00	.00	.00	.00	.00	.00	.00	.00	.67	.91	.88	.85	.84	.84	.84	.84	.84	.84	.85	.87	.89
5	.00	.00	.00	.00	.00	.00	.00	.67	.91	.88	.84	.83	.83	.83	.83	.83	.83	.84	.85	.87	.89
6	.00	.00	.00	.00	.00	.00	.63	.91	.88	.84	.83	.83	.83	.83	.83	.83	.83	.83	.85	.87	.89
7	.00	.00	.00	.00	.00	.63	.91	.88	.84	.82	.82	.83	.83	.82	.82	.81	.81	.82	.84	.87	.89
8	.00	.00	.00	.00	.63	.91	.88	.84	.82	.82	.82	.82	.82	.78	.79	.76	.79	.81	.84	.87	.89
9	.00	.00	.00	.63	.91	.88	.84	.82	.82	.82	.81	.80	.75	.66	.66	.70	.78	.81	.84	.87	.89
10	.00	.00	.63	.91	.88	.84	.82	.82	.82	.82	.79	.60	.47	.45	.60	.68	.78	.81	.84	.87	.89
11	.00	.63	.91	.88	.84	.82	.82	.82	.81	.67	.47	.39	.47	.43	.58	.68	.78	.81	.84	.87	.89
12	.63	.91	.88	.84	.82	.82	.81	.81	.63	.45	.41	.33	.36	.44	.58	.68	.78	.81	.84	.87	.89
13	.91	.88	.84	.82	.82	.81	.79	.64	.55	.39	.37	.36	.36	.44	.58	.68	.78	.81	.84	.87	.89
14	.88	.84	.82	.82	.81	.81	.61	.42	.32	.27	.37	.37	.36	.44	.58	.68	.78	.81	.84	.87	.89
15	.84	.82	.82	.81	.81	.54	.44	.24	.24	.25	.38	.37	.36	.44	.58	.68	.78	.81	.84	.87	.89

In Table 2 we can see the precision values for each set of hyper-parameters produced by the grid search. Cells coloured red have the lowest precision values, yellow cells have a higher precision value and green cells have the highest precision.

Table 3: Recall values from Hyper-Parameters Grid Search

C Exponent \ Gamma Exponent	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
-5	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
-4	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
-3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.06	.20	.21	.10
-2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.15	.41	.53	.53	.40
-1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.19	.47	.62	.67	.65	.56
0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.19	.48	.64	.73	.74	.71	.65
1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.02	.18	.46	.62	.74	.79	.78	.74	.67
2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.02	.17	.43	.60	.73	.80	.82	.79	.74	.67
3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.02	.16	.41	.57	.70	.78	.82	.82	.79	.73	.67
4	.00	.00	.00	.00	.00	.00	.00	.00	.00	.02	.15	.39	.55	.67	.76	.82	.83	.82	.78	.73	.66
5	.00	.00	.00	.00	.00	.00	.00	.00	.02	.15	.39	.53	.64	.73	.80	.83	.83	.81	.77	.73	.66
6	.00	.00	.00	.00	.00	.00	.00	.02	.15	.38	.52	.62	.71	.78	.82	.83	.82	.79	.77	.72	.66
7	.00	.00	.00	.00	.00	.00	.02	.14	.38	.51	.60	.68	.76	.81	.83	.82	.80	.78	.76	.72	.66
8	.00	.00	.00	.00	.00	.02	.14	.37	.51	.59	.67	.73	.79	.81	.81	.79	.78	.78	.76	.72	.66
9	.00	.00	.00	.00	.02	.14	.37	.50	.59	.66	.72	.77	.76	.74	.74	.75	.77	.77	.76	.72	.66
10	.00	.00	.00	.02	.14	.37	.50	.58	.65	.71	.74	.71	.68	.66	.70	.74	.77	.77	.76	.72	.66
11	.00	.00	.02	.14	.37	.50	.58	.65	.70	.72	.64	.56	.59	.63	.71	.74	.77	.77	.76	.72	.66
12	.00	.02	.14	.37	.50	.58	.64	.70	.69	.64	.50	.56	.56	.62	.71	.74	.77	.77	.76	.72	.66
13	.02	.14	.37	.50	.58	.64	.69	.68	.56	.52	.46	.56	.56	.62	.71	.74	.77	.77	.76	.72	.66
14	.14	.37	.50	.58	.64	.69	.69	.55	.44	.45	.45	.56	.56	.62	.71	.74	.77	.77	.76	.72	.66
15	.37	.50	.58	.64	.68	.66	.57	.46	.39	.42	.45	.56	.56	.62	.71	.74	.77	.77	.76	.72	.66

In Table 3 we can see the recall values for each set of hyper-parameters found by the grid search. The colouring scheme is the same used in Table 2.

Table 4: F-Score values from Hyper-Parameters Grid Search

C Exponent \ Gamma Exponent	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
-5	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
-4	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00
-3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.11	.33	.34	.17
-2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.01	.25	.56	.66	.66	.55
-1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.02	.31	.61	.72	.75	.74	.68
0	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.03	.32	.62	.73	.79	.79	.78	.75
1	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.03	.30	.60	.72	.79	.82	.82	.80	.76
2	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.03	.28	.57	.70	.78	.82	.83	.82	.80	.76
3	.00	.00	.00	.00	.00	.00	.00	.00	.00	.00	.03	.27	.55	.68	.76	.81	.83	.83	.82	.79	.76
4	.00	.00	.00	.00	.00	.00	.00	.00	.00	.03	.26	.54	.66	.74	.80	.83	.84	.83	.81	.79	.76
5	.00	.00	.00	.00	.00	.00	.00	.00	.03	.25	.53	.65	.72	.78	.82	.83	.83	.82	.81	.79	.76
6	.00	.00	.00	.00	.00	.00	.00	.03	.25	.52	.64	.71	.76	.80	.83	.83	.82	.81	.80	.79	.76
7	.00	.00	.00	.00	.00	.00	.03	.25	.52	.63	.70	.75	.79	.82	.82	.82	.81	.80	.80	.79	.76
8	.00	.00	.00	.00	.00	.03	.25	.52	.63	.69	.74	.78	.80	.79	.80	.77	.79	.79	.80	.79	.76
9	.00	.00	.00	.00	.03	.25	.52	.62	.68	.73	.76	.78	.76	.70	.70	.72	.77	.79	.80	.79	.76
10	.00	.00	.00	.03	.25	.52	.62	.68	.72	.76	.76	.65	.55	.54	.64	.71	.77	.79	.80	.79	.76
11	.00	.00	.03	.25	.52	.62	.68	.72	.75	.69	.54	.46	.52	.51	.64	.71	.77	.79	.80	.79	.76
12	.00	.03	.25	.52	.62	.68	.72	.75	.66	.53	.45	.41	.44	.52	.64	.71	.77	.79	.80	.79	.76
13	.03	.25	.52	.62	.68	.72	.74	.66	.56	.44	.41	.44	.44	.52	.64	.71	.77	.79	.80	.79	.76
14	.25	.52	.62	.68	.72	.75	.65	.47	.37	.33	.40	.44	.44	.52	.64	.71	.77	.79	.80	.79	.76
15	.52	.62	.68	.71	.74	.60	.49	.32	.30	.32	.41	.44	.44	.52	.64	.71	.77	.79	.80	.79	.76

In Table 4 we can see the F-Score for each set of hyper-parameters calculated from the precision (Table 2) and recall (Table 3) values. We can see the precision values along the leading edge (diagonal green line from bottom left to top right) are the largest found, however when we compare these values with their corresponding recall values, we see that the classifier was achieving very low recall levels. Both the precision and recall values were lower in the bottom-middle area of the table. When we examine the F-Score values we can see that the optimal hyper-parameters were located in the middle right portion of the table.

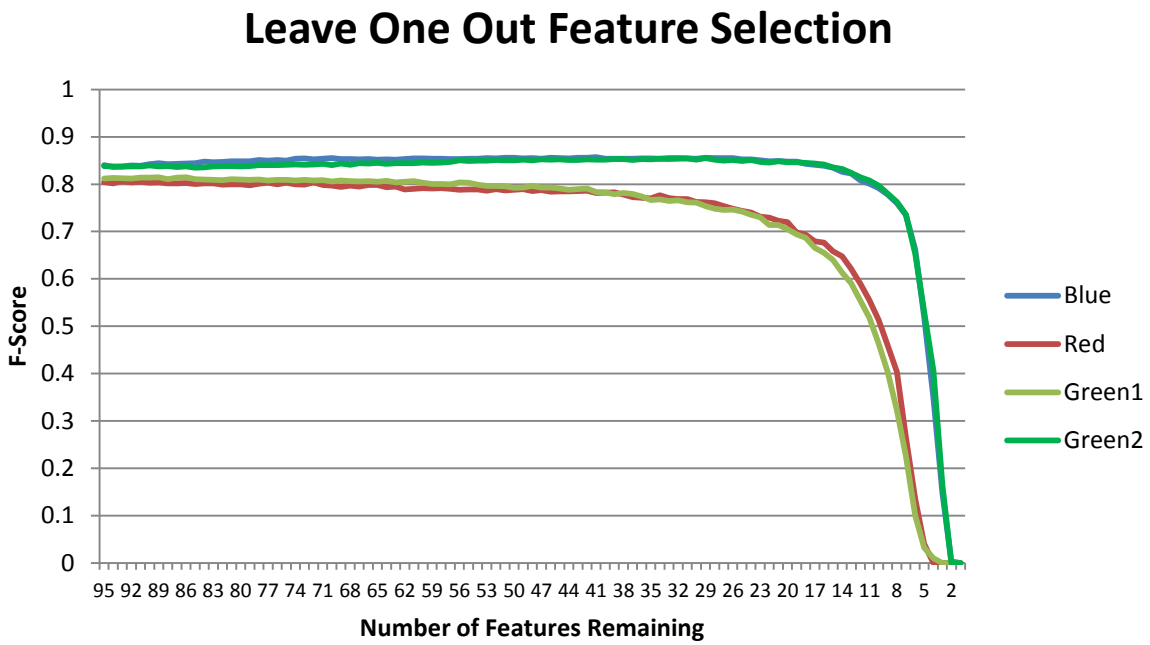


Figure 18: F-Score for the *leave one out* method for feature selection

In Figure 18 we can see the F-Score of the four SVMs during the *leave one out* feature selection process. We can that the Blue and Green2 SVMs maintained a high level of performance until around 10 features remained where the performance rapidly dropped. The Red and Green1 SVMs had a significantly lower performance than the other two SVMs; their performance started falling

much sooner and gradually. The blue and green2 SVMs initially increased in performance as features are removed before plateauing and finally falling off.

Table 5: Comparison of Standard Deviation between Whole Image Classification and 10-Fold validation over 20 trials

	Whole Image Classification	10-Fold
Standard Deviation	0.019799	9.88E-06

In Table 5 we can see the Standard Deviation values for the two methods developed for measuring the performance of the SVMs. We can see that the standard deviation for the whole image classification method is approximately 2000 times greater than that of the 10-fold cross validation method. This experiment was performed in order to find the method for measuring the performance of the classifier with the least variability.

1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	0
0	0	1	1	0	0	0	1
0	0	1	1	1	0	0	1
2	0	2	1	1	0	1	1
0	0	1	1	1	0	0	0
1	1	0	2	0	0	1	1
0	0	1	1	1	0	0	0

Figure 19: Blue feature mask

1	2	1	2	0	1	0	2
1	1	2	0	0	0	0	0
1	1	1	0	1	0	1	0
2	0	1	1	0	0	0	1
0	1	0	1	0	1	0	1
1	1	1	0	0	0	0	0
0	1	0	1	0	2	0	1
1	1	1	0	0	0	0	0

Figure 20: Green1 feature mask

0	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0
1	1	1	0	0	0	1	0
1	1	0	1	0	0	0	0
0	1	0	0	0	0	0	0
2	1	0	1	0	1	0	0
0	2	0	1	0	1	1	2
1	2	0	2	0	1	0	1

Figure 21: Red feature mask

0	0	1	1	0	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	0
1	0	1	0	1	0	0	0
1	0	1	2	0	1	0	0
0	0	1	0	1	0	0	0
1	1	2	1	2	0	1	0
0	0	1	1	1	0	1	0

Figure 22: Green2 feature mask

In Figures (Figure 19, Figure 20, Figure 21, Figure 22) we can see which pixels, relative to the pixel of interest, are being used for classification by each SVM. Red cells show the relative position of pixels which are not being used for classification, yellow are pixels that are being used once, and green are pixels being used twice. Pixels can be used twice if they are included in more than one feature such as a row, column, or individual pixel. It is possible for a pixel to be used three times, however once only 30 features remained, there were no instances where this occurred. In these images we can see some structure to the remaining features, although the structure does vary between the feature masks. All four feature masks have a somewhat uniform distribution vertically but vary substantially in their horizontal distribution.

Table 6: Precision values from Examples Grid Search

Wire Examples (hundreds) \ Non-Wire Examples (hundreds)	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
1	.248	.242	.241	.237	.231	.229	.232	.226	.222	.212	.223	.215	.209	.205	.189	.219	.205	.204	.208	.195	.208	.204	.206	.208	.207
2	.198	.242	.237	.240	.240	.240	.236	.234	.232	.232	.230	.231	.230	.228	.229	.222	.217	.231	.216	.232	.226	.221	.221	.230	.220
3	.212	.247	.246	.245	.243	.239	.239	.238	.239	.236	.234	.233	.236	.230	.230	.231	.230	.228	.227	.233	.229	.229	.229	.229	.229
4	.220	.246	.245	.242	.245	.243	.241	.240	.241	.242	.238	.237	.235	.237	.235	.236	.236	.234	.233	.235	.234	.233	.233	.232	.235
5	.225	.246	.245	.247	.243	.244	.242	.242	.244	.242	.241	.242	.238	.240	.238	.236	.241	.236	.239	.237	.234	.237	.237	.236	.237
6	.230	.223	.245	.245	.245	.245	.243	.245	.243	.245	.240	.241	.240	.241	.240	.240	.241	.239	.241	.240	.239	.237	.239	.235	.236
7	.230	.215	.246	.246	.246	.246	.245	.245	.244	.244	.241	.242	.245	.241	.241	.240	.239	.241	.239	.241	.238	.240	.239	.236	.240
8	.235	.214	.246	.246	.247	.246	.245	.245	.244	.244	.245	.243	.244	.244	.243	.241	.240	.241	.240	.242	.239	.240	.243	.242	.240
9	.235	.218	.246	.247	.247	.245	.245	.246	.245	.245	.243	.245	.243	.242	.241	.242	.240	.242	.240	.243	.242	.242	.241	.240	.241
10	.237	.220	.245	.245	.247	.246	.245	.247	.247	.246	.244	.244	.245	.244	.243	.245	.243	.243	.240	.242	.242	.242	.240	.241	.241
11	.238	.222	.248	.248	.247	.247	.246	.247	.246	.246	.245	.244	.245	.244	.244	.245	.243	.244	.244	.243	.242	.242	.242	.244	.243
12	.239	.223	.247	.247	.248	.247	.246	.247	.246	.247	.245	.245	.245	.244	.244	.245	.245	.244	.244	.244	.244	.243	.244	.242	.243
13	.240	.228	.240	.249	.247	.248	.247	.247	.247	.247	.245	.246	.246	.245	.244	.245	.243	.244	.245	.243	.244	.244	.244	.243	.243
14	.240	.227	.225	.248	.247	.248	.248	.247	.247	.246	.247	.247	.245	.245	.245	.245	.245	.244	.244	.245	.245	.244	.244	.244	.245
15	.241	.228	.225	.249	.249	.248	.248	.247	.247	.247	.246	.246	.246	.246	.246	.246	.246	.244	.246	.245	.244	.245	.245	.244	.245

Table 7: Recall values from Examples Grid Search

Wire Examples (hundreds) \ Non-Wire Examples (hundreds)	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
1	.177	.160	.141	.134	.154	.123	.144	.140	.131	.139	.145	.116	.120	.133	.109	.141	.125	.118	.116	.103	.105	.138	.124	.106	.126
2	.250	.179	.177	.181	.173	.158	.155	.166	.173	.171	.162	.157	.168	.156	.166	.153	.159	.178	.158	.166	.166	.157	.159	.166	.169
3	.249	.192	.187	.180	.180	.182	.179	.178	.182	.175	.172	.175	.173	.170	.178	.173	.171	.171	.175	.179	.169	.172	.178	.167	.178
4	.250	.208	.198	.192	.195	.191	.193	.192	.195	.190	.188	.188	.191	.188	.188	.185	.188	.188	.187	.181	.186	.189	.180	.185	.186
5	.249	.218	.205	.202	.199	.200	.198	.197	.198	.200	.197	.198	.194	.199	.194	.192	.193	.194	.190	.195	.190	.197	.188	.194	.194
6	.250	.237	.212	.205	.208	.209	.204	.202	.202	.206	.201	.206	.201	.203	.205	.203	.206	.204	.205	.207	.203	.201	.206	.203	.207
7	.250	.249	.220	.218	.210	.206	.212	.212	.211	.210	.213	.208	.206	.207	.207	.215	.212	.212	.204	.208	.211	.208	.206	.214	.212
8	.250	.250	.224	.216	.219	.216	.216	.216	.215	.213	.213	.213	.213	.215	.215	.216	.215	.214	.210	.211	.208	.215	.215	.215	.212
9	.250	.250	.228	.218	.225	.222	.217	.223	.217	.214	.220	.221	.217	.217	.217	.216	.220	.219	.214	.216	.219	.214	.217	.217	.219
10	.250	.250	.234	.223	.222	.225	.226	.220	.223	.221	.225	.223	.225	.224	.222	.222	.220	.218	.222	.223	.221	.222	.220	.217	.221
11	.250	.250	.238	.232	.227	.227	.227	.227	.225	.226	.224	.225	.225	.218	.226	.225	.227	.223	.224	.227	.222	.228	.225	.222	.225
12	.250	.250	.236	.234	.233	.228	.225	.231	.230	.226	.227	.229	.229	.228	.229	.224	.228	.229	.229	.226	.225	.226	.228	.229	.228
13	.250	.250	.245	.232	.230	.232	.230	.235	.231	.230	.230	.232	.231	.230	.232	.228	.226	.228	.229	.228	.227	.229	.229	.230	.229
14	.250	.250	.250	.236	.236	.234	.232	.235	.231	.233	.234	.234	.235	.234	.233	.230	.235	.232	.232	.231	.233	.233	.233	.235	.232
15	.250	.250	.250	.239	.235	.234	.237	.237	.236	.236	.236	.237	.234	.235	.237	.235	.235	.231	.233	.237	.235	.236	.235	.235	.232

Table 8: F-Score values from Examples Grid Search

Wire Examples (hundreds) \ Non-Wire Examples (hundreds)	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
1	.207	.193	.178	.171	.185	.160	.177	.173	.165	.168	.176	.151	.152	.161	.138	.172	.155	.149	.149	.134	.140	.164	.155	.141	.157
2	.221	.206	.202	.206	.201	.190	.187	.194	.198	.197	.190	.187	.194	.185	.192	.181	.183	.201	.182	.193	.191	.183	.185	.193	.191
3	.229	.216	.213	.207	.207	.207	.205	.204	.206	.201	.198	.200	.199	.196	.200	.198	.196	.196	.197	.202	.194	.196	.200	.193	.200
4	.234	.226	.219	.214	.217	.214	.214	.213	.216	.213	.210	.210	.211	.210	.209	.207	.209	.208	.207	.204	.207	.209	.203	.206	.207
5	.237	.231	.223	.222	.219	.219	.218	.217	.219	.219	.217	.218	.214	.217	.213	.211	.214	.213	.212	.214	.210	.215	.209	.213	.213
6	.239	.230	.227	.223	.225	.225	.222	.221	.221	.224	.219	.222	.219	.220	.221	.220	.222	.220	.221	.222	.219	.217	.221	.218	.221
7	.240	.231	.232	.231	.227	.224	.227	.228	.227	.226	.226	.223	.224	.223	.223	.227	.225	.225	.220	.223	.223	.223	.221	.224	.225
8	.242	.230	.234	.230	.232	.230	.229	.230	.229	.227	.228	.227	.228	.228	.228	.228	.226	.226	.224	.225	.222	.227	.228	.228	.225
9	.242	.233	.237	.231	.236	.233	.230	.234	.230	.228	.231	.232	.229	.229	.228	.228	.230	.230	.226	.229	.230	.227	.228	.228	.229
10	.243	.234	.240	.234	.234	.235	.235	.233	.234	.233	.234	.233	.235	.234	.232	.233	.231	.230	.231	.232	.231	.232	.230	.228	.230
11	.244	.235	.243	.240	.237	.236	.236	.236	.235	.236	.234	.234	.234	.230	.234	.234	.235	.233	.234	.235	.232	.235	.233	.232	.233
12	.244	.235	.242	.240	.240	.237	.235	.238	.238	.236	.236	.237	.237	.236	.236	.234	.236	.236	.236	.235	.234	.234	.236	.235	.235
13	.245	.238	.242	.240	.238	.240	.238	.241	.239	.238	.238	.239	.238	.237	.238	.236	.235	.236	.237	.235	.235	.236	.236	.236	.236
14	.245	.238	.237	.242	.241	.241	.240	.241	.238	.239	.240	.240	.240	.240	.239	.237	.240	.238	.238	.238	.239	.238	.238	.239	.238
15	.245	.239	.236	.244	.242	.241	.242	.242	.241	.241	.241	.242	.240	.240	.242	.240	.240	.238	.239	.241	.239	.240	.240	.240	.238

Table 6 shows the relationship between the classifier's precision and the amount of positive and negative training examples used. This relationship shows that having significantly more positive than negative, or negative than positive, results in the classifier achieving lower levels of performance.

Table 7 shows that once the number of negative examples used by the classifier is greater than 500, further increasing it has a minimal effect on the recall ability of the classifier. Instead the recall ability is mostly dependent on the number of positive examples used for training. We can see that the classifier achieves an excellent level of recall with less than 500 negative examples. This also corresponds to a reduced level of precision seen in Table 6. The reduced number of negative training examples results in the classifier making a larger number of positive classifications; reducing the precision and increasing the recall.

Table 8 shows the resulting F-Scores from the Example Search. We can see that the overall performance of the classifier is largely dependent on the number of positive examples, and less on the number of negative examples. The classifier achieves the highest F-Scores when the number of negative numbers is smaller. We are primarily interested in increasing the precision of the classifier and secondarily in the recall, while the F-Score evenly weights both of these values. For this reason we would be more inclined to look at the individual Recall and Precision results rather than the combined F-Score for analysing this experiment.

Table 9: Comparison of classification with different feature sets

	Selected Features			Row/Col Features			ALL Features			Raw Features		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
Image 1	.891	.055	1.65	.950	.160	1.90	.952	.225	5.03	.951	.218	2.91
Image 2	.887	.053	1.72	.939	.137	1.98	.938	.208	5.08	.936	.196	3.01
Image 3	.930	.065	1.73	.956	.186	1.90	.934	.227	5.10	.939	.223	3.04
Image 4	.885	.043	1.69	.908	.124	1.90	.912	.190	5.01	.915	.183	3.01
Image 5	.878	.043	1.67	.919	.119	1.86	.926	.181	5.00	.927	.171	3.03
Image 6	.786	.043	1.69	.894	.128	1.88	.927	.187	5.02	.923	.178	3.04
Image 7	.909	.053	1.69	.943	.178	1.90	.933	.232	5.03	.938	.228	3.06
Image 8	.869	.052	1.67	.973	.159	1.87	.967	.223	5.00	.970	.217	3.01
Mean	**.879**	**.051**	**1.69**	**.935**	**.149**	**1.90**	**.936**	**.209**	**5.03**	**.937**	**.202**	**3.01**

In Table 9 we can see the overall performance of the classifier for different feature sets. We can see that when using the 20 best *selected features* the classifier achieved a mean precision of 0.88 and recall of 0.051 which is significantly lower than the three other feature sets, however it did perform the classification in less time; only 1.7 seconds. The *Row/Col*, *All 96*, and *Raw features* sets all achieved similar precision scores. *The Row/Col feature* set achieved a smaller recall value than using *all 96 features* and the *raw feature set*. The *selected feature set* consisted of 20 features, the *Row/Col feature set* consisted of 32 features, and feature set which contained *all of the features* in the features library had 96 total features. We can see that these 4 feature sets took a classification time proportional to the amount of features they used. The recall values achieved are smaller than what might be expected from a SVM, this is due in part to the method used for measuring performance; which is done pixel by pixel. Over an entire wire, having 5% of the pixels found is enough to classify the wire, if a high level of precision is achieved (see Figure 25 which achieved 7% recall).

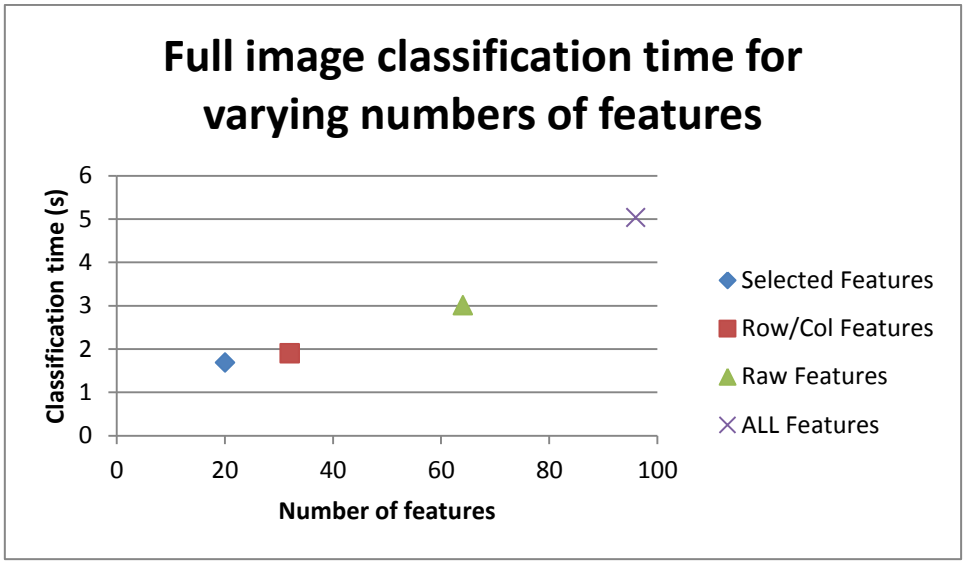


Figure 23: Time required by the vectorisation and classification portion of the system for varying numbers of features on a 960 by 1280 pixel image

In Figure 23 we can clearly see the relationship between the number of features in a vector and the amount of time required for the classification to be completed. The relationship appears to be asymptotically linear.

Figure 24: All 96 features (precision 0.94, recall 0.22)

Figure 25: 20 Best Features (precision 0.93, recall 0.07)

Figure 24 and Figure 25 visually show the difference between the different feature sets used for classification. These images show a sub-region of the images used for measuring the performance of the classifiers. We can see that using all 96 features gives a much higher rate of recall than using the 20 best features found. In this comparison we can see that both classifiers have achieved a similar level of precision, however in other images this in not the case (Table 9).

Discussion

Hyper-Parameters

Choosing the optimal hyper-parameters was key to getting the highest performance from the classifier. Using a grid search allowed us to visualise the possible search space and choose the best values for C and Gamma. This search shows the relationship between C and Gamma and how varying these parameters could affect the precision and recall achieved by our classifier. By using the F-Score we are able to select a set of hyper-parameters that achieve both good precision and recall values. We can see that the optimal hyper-parameters were C = 2^4 and Gamma = 2^6. We can generally sacrifice recall to increase precision so long as enough positive classifications can be made to still fit a wire; increasing the precision means there are less false positives which will make subsequent tasks, like wire fitting, easier.

Feature Selection

The feature selection process is required to reduce the number of features while still maintaining an acceptable level of performance. Fewer features results in less computation when creating each vector and by the SVM in order to classify each example. While performing the *leave one out* method of feature selection we observed some interesting behaviour. Two of the support vector machines diverged from the other two, while each pair of SVM maintained a very similar level of performance throughout the feature selection process. The blue and green2 SVMs both initially showed slight increases in performance as features were being removed. This can be attributed to reducing the amount of over fitting done by the classifier which is often caused by having redundant or useless features in the training data. We then see the performance plateaued until only 20 features remain; at this point the performance begins a rapid decline which is to be expected as the classifier starts to starve for features and can no longer maintain a high level of performance. The red and green1 SVMs showed a different performance profile as features were removed, neither showed the initial increase in performance that was observed in the blue and green2 SVMs. Once only 60 features remained, both red and green1 SVMs began their decline in performance; some 20 features earlier than in the other two SVMs. At 30 remaining features, both red and green1 SVMs had begun rapid decline in performance, approximately 10 features earlier than the other two SVMs. It is unknown what would cause the divergence in performance between the classifiers however it is worth noting that both the red and green1 SVMs fell on the same row while the blue and green2 SVMs fell on the row below (see Figure 13). Additional testing would need to be done in order to discover the course of this divergence; all four SVMs were given access to the same feature library and trained under the same set of conditions.

Training Examples

Selecting the optimal set of training examples is critical for allowing the SVMs to correctly calibrate the misclassification costs. As can be seen in Table 8, selecting too few positive examples results in the decreased performance of the classifier. Not selecting enough Negative examples results in a larger number of false positives and a decreased precision. Not selecting enough positive examples results in a reduced recall rate which can be seen in Table 8; the F-Scores only begin to reach optimal values after 400 positive examples are included in the training data.

Overall Performance

Given the optimal hyper-parameters and number of training examples found previously, we tested the performance of the classifier using four different feature sets. Three of the four feature sets used the pixel masking system previously discussed; these were the 20 optimal features for each SVM, the 32 alternating rows and columns, and the full set of 96 features. The fourth feature set contained 64 features; one for each pixel. This final feature set did not use a mask based vectoriser like the other three feature sets, instead it used a vectoriser which directly iterated over a region of interest centred on the pixel being classified.

Classification Time

The time taking to classify an entire image increases in relation to the number of features used in each vector. This polynomial growth in performance is to be expected because the underlying SVM implementation scales between $O\left(n_{features} \times n_{samples}^{2}\right)$ and $O\left(n_{features} \times n_{samples}^{3}\right)$ (18). We were expecting the Raw Features experiment to fall slightly off this trend because its iterative method of vectorisation is more efficient than the mask based method. The mask based method requires that the vectoriser iterates over 64 Boolean values for each feature, then perform a comparison and fetch a value if the given pixel is part of the current feature, while the Raw feature set only iterates over the 64 pixels once, creating features out of each pixel. However this difference in performance was not observed because the time taken for vectorisation was negligible in comparison to the time required for classification. The results obtained reiterate the importance of reducing the size of the feature set in order for the system to be able to classify examples at an acceptable rate. While even the fastest of the experiments, taking 1.69 seconds, is not fast enough for deployment. The classification time could be significantly reduced by pre-processing the images with a fast heuristic classifier, providing a level of boosting by removing pixels with a low likelihood of belonging to a wire; such as pixels that fall on the blue backdrop.

Recall

We see a significant decrease in the recall of the classifiers as the feature sets are reduced. All 96 features and the Raw 64 features both achieved high recall values of 0.21 and 0.20 respectively. The 96 features did not achieve a significantly greater recall value than the raw 64 features. This is because there is no additional information as the 32 additional features are derived from the original 64. When using the row/col feature set we observed a significant drop in recall down to 0.15. The recall continues falling as we reduce the size of the feature set. With the 20 best features remaining we have the lowest recall of 0.05. This relationship shows that the recall drops significantly as the size of the feature set decreases. However achieving a recall rate of 0.05 still allows for the accurate identification of wires; this can be observed by comparing Figure 24 and Figure 25.

Precision

The precision value maintains a value between 0.93 and 0.94 for the experiments with the larger features sets, row/col, all 96, and raw. The 20 best selected features did have a reduced precision of 0.88, which is a significant difference from the other three feature sets. While the precision level of 0.88 is significantly lower than what was achieved by the other feature sets, this level of precision should still be great enough to allow for the wires' locations to be found; as seen by comparing Figure 24 and Figure 25.

Improving Classification Time

The camera on the robot captures 7.5 frames each second, this gives the system 133ms to perform all of the classification and decision making tasks. The wire detection system is not required to operate at this speed because most of the content in one frame will still be visible in the next. It would make integration into the system easier if it could perform at this speed.

By finding the optimal subset of features used by the classifier, we have decreased the amount of time required for classification. However to achieve a significant improvement in performance we need to find more effective means of optimising the classifier. One such way of reducing the computation required would be to use a pre-processing heuristic. This pre-processor for example could mask out all of the pixels belonging to the blue background. The accurate classifier driven by the techniques discussed here would then only attempt to classify the remaining pixels. In some images more than 90% of the pixels belonged to the background so this technique alone could achieve a significant boost in performance.

Conclusion

We have developed a system for training a classifier ideally suited to classifying wires from images of the canopy of grape vines. Our proposed classifier is not required to be tuned and uses no heuristics to identify wires. Instead we use a set of training data to train the system how to accurately identify wires. This frees the end user from the burden of manipulating parameters in order to achieve the desired results. The system only requires that the user provides an image containing the locations of the wires and allow the system to learn how to identify them.

We have been able to achieve precision and recall values great enough to allow wires to be fitted to the classified pixels. These classified pixels can then be fed into the wire fitting system to find a wire model which represents the wire locations.

Future work

Ultimately the classifier presented would be only a portion of a system designed to extract the wire locations. By applying pre-processing techniques such as the removal of noise or large scale objects from the image, the task performed by the classifier can be simplified. Post-processing techniques such as the N out of M test used in our previous research (wire detection with neural networks) could be used to reduce the number of false positives.

The cause of the divergence observed in the performance between the two sets of SVMs during the feature selection process remains undiscovered. One explanation is the eight by eight region of pixels was unable to constantly have good features created from it depending on the pixel colour it was centred on. To test this theory the use of a larger region of interest, perhaps ten by ten pixels, could be used to eliminate this effect.

We found four separate optimal subsets of features; one for each SVM. We did not however find individual hyper-parameters for each SVM. It was assumed that finding individual sets of hyper-parameters would result in similar values being found as the SVMs were working in the same problem space. However some additional performance may be obtainable by treating the hyper-parameters separately for each SVM.

The vines belonging to a vineyard will not all be homogeneous, instead plant structure will vary based on soil quality, irrigation, frost, etc.; all of these can vary across a single vineyard. The varying plant structure will affect the plant canopy and ultimately the environment the wires are being extracted from. In order to allow the system to adapt to slow changes across a vineyard we propose the development of a dynamic training system. This would allow the system to be constantly learning and adapting to variations in the input data.

Bibliography

1. **Trent Ball, Raymond J. Folwell, Jack Watson, Markus Keller.** *Establishment and Annual Production Costs for Washington Concord Grapes.* Washington : WSU Extension, 2004.

2. *Construction a Vineyard Trellis.* **Domoto, Paul.** s.l. : Iowa State University, 2002. Iowa Grape Growers Conference.

3. **Dennis J. Yelton, Robert L. Wright.** *System and Method for Passive Wire Detection. 7,512,258 B2* United Stated of America, 19 July 2005.

4. *A Fast and Robust Approach to Lane Marking Detection and Lane Tracking.* **Christian Lipski, Bjorn Scholz, Kai Berger, Christian Linz, Timo Stich.** Santa Fe : s.n., 2008. Southwest Symposium on Image Analysis and Interpretation.

5. *Detecting Structured Light Patterns in Colour Images using a Support Vector Machine.* **T. Botterill, R. Green, S. Mills.** Paris : s.n., 2013. International Conference on Image Processing.

6. *An empirical comparison of supervised learning algorithms.* **Rich Caruana, Alexandru Niculescu.** Pittsburgh : s.n., 2006. International Conference on Machine Learning.

7. *The support vector machine under test.* **David Meyera, FriedrichLeisch , Kurt Hornik.** 1-2, 2003, Neurocomputing, Vol. 55, pp. 169-186.

8. *New Support Vector Algorithms.* **B. Scholkopf, A. J. Smola, R. C. Williamson, P. L. Bartlett.** 12, 2000, Neural Computation, pp. 1207-1245.

9. *A neural network approach to robust shape classification.* **Lalit Gupta, Mohammad R. Sayeh, Ravi Tammana.** 8, 1989, Pattern Recognition, Vol. 23, pp. 563-568.

10. *Detecting Wires in the Canopy of Grapevines using Neural Networks.* **Joshua McCulloch, Richard Green.** Wellington : s.n., 2013. Image and Vision Computing New Zealand.

11. *Multiscale Edge Detection and Fiber Enhancement Using Differences of Oriented Means.* **Meirac Galun, Ronen Basri, Achi Brandt.** Rio de Janeiro : s.n., 2007. International Conference on Computer Vision.

12. **Flowers, Simon.** *Low-level Image Segmentation for a Vine Imaging Robot.* Canterbury : University of Canterbury, 2012.

13. **Bayer, Bryce E.** *Color Imaging Array. 555,477* United States of America, 5 March 1975.

14. Demosaicing Algorithms for Digital Cameras. [Online] ImageVal. [Cited: 1 7 2013.] http://www.imageval.com/public/Products/ISET/ISET_Manual/Demosaicing.htm.

15. *Quantitative analysis of skeletonisation algorithms for modelling of branches.* **Tom Botterill, Will Gittoes, Richard Green.** Auckland : s.n., 2011. Image and Vision Computing New Zealand.

16. *Reconstructing partially visible models using stereo vision , structured light , and the g2o framework.* **Tom Botterill, Richard Green, Steven Mills.** Dunedin : s.n., 2012. IVCNZ' 12. pp. 370-375.

17. *A decision-theoretic formulation for sparse stereo correspondence problems.* **Tom Botterill, Richard Green.** Sydney : s.n., 2013. International Conference on Computer Vision.

18. Support Vector Machines. *Scikit Learn.* [Online] 2010. [Cited: 9 November 2013.] http://scikit-learn.org/stable/modules/svm.html.

19. **Deng, Kan.** OMEGA: On-line Memory Based General Purpose System Classifier. Pittsburgh : Carnegie Mellon University, 1998, pp. 117-132.

20. *A Direct Method of Nonparametric Measurement Selection.* **Whitney, A. Wayne.** 9, 2006, Transactions on Computers, Vols. C-20, pp. 1100-1103.

21. *Evaluation of simple performance measures for tuning SVM hyperparameters.* **Kaibo Duan, S. Sathiya Keerthi, Aun Neow Poo.** 51, 2003, Neurocomputing, pp. 41-59.

22. **Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin.** *A Practical Guide to Support Vector Classification.* Taipei : National Taiwan University, 2003.

Appendices

Example of Training Images

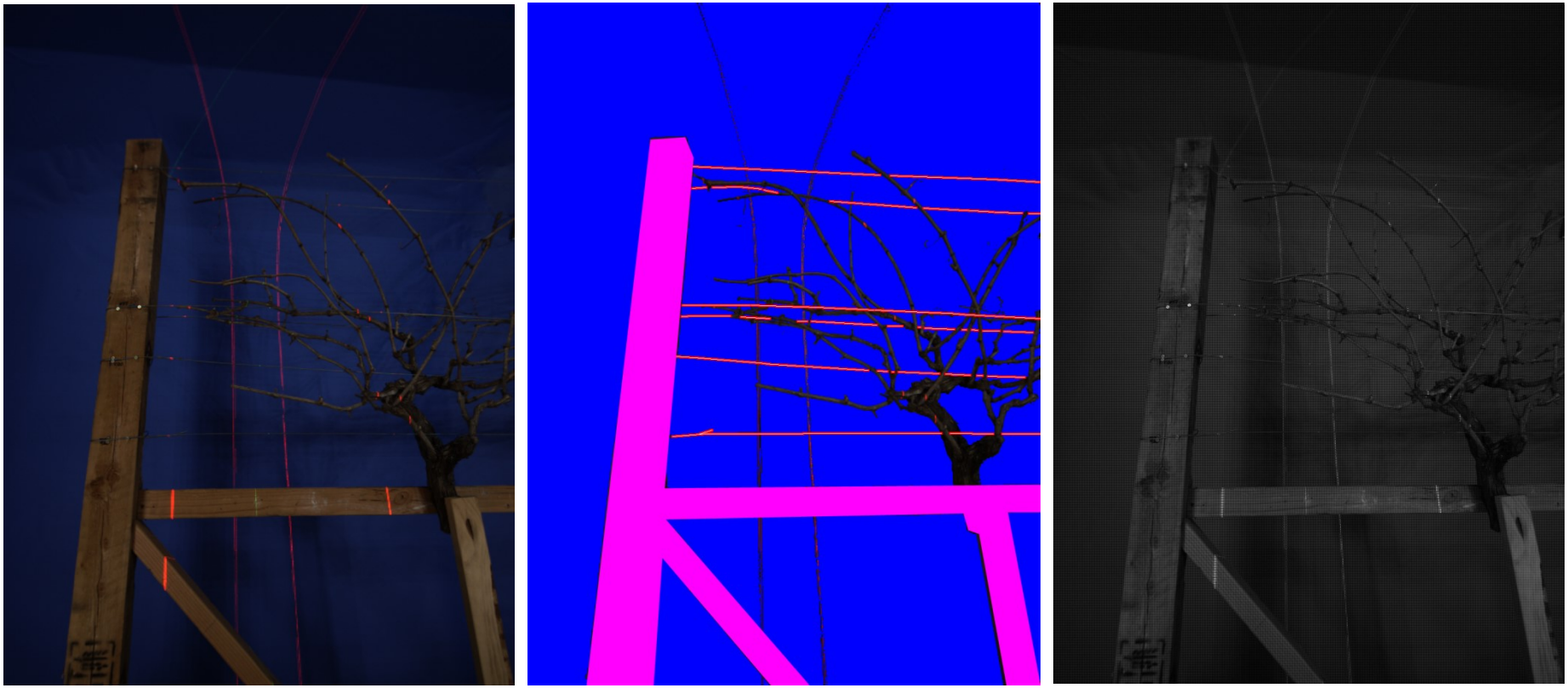
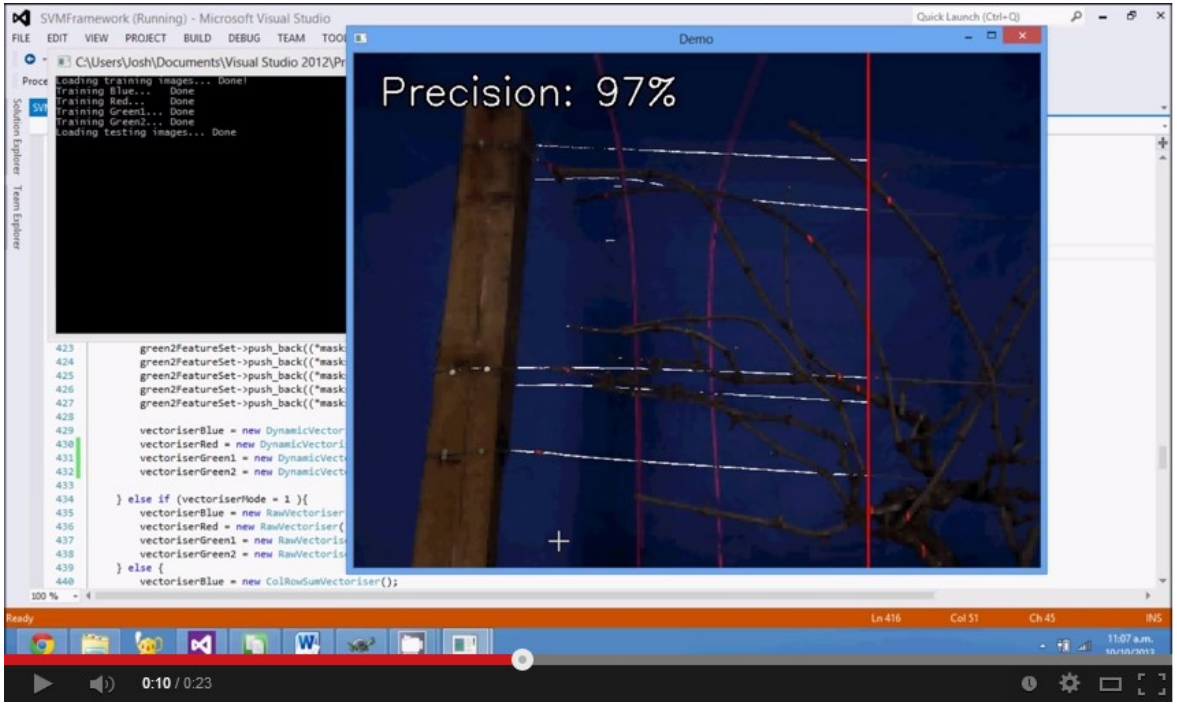


Figure 26: Colour, Marked, and Bayer versions of training data

Video Demonstration



http://www.youtube.com/watch?v=_yFVTFVCrWI