Automatic Inventory Identification through Image Recognition

by
Matt Triff

Thesis submitted to Saint Mary's University
in partial fulfilment of the requirements for the
Degree of Master of Science in Computing and Data Analytics

July, 2017, Halifax, Nova Scotia

Approved: Dr. Hai Wang
Supervisor
Department of Finance, Info Systems
and Management Science

Approved: Dr. Haiyi Zhang
External Examiner
Acadia University

Approved: Dr. Yasushi Akiyama
Supervisory Committee Member
Department of Mathematics
and Computing Science

Date: July 16, 2017

Abstract

"Automatic Inventory Identification through Image Recognition"

by Matt Triff

Abstract: Companies are looking to use technology to replace repetitive, low skill tasks currently performed by human employees. One such series of tasks is conducting an inventory of items within a warehouse. This requires employees to identify individual items on a shelf. This task is well-suited to the application of computer vision techniques. Images of shelves in a warehouse can be acquired and analyzed. The goal of this research is to accurately identify items within the images for the purpose of inventory management.

This objective requires both the detection of objects within an image, and the identification of those objects. I propose a series of techniques to identify an item against a set of template images of potential items. To demonstrate the effectiveness of these techniques, I have developed a prototype system that performs an inventory of equipment in a data centre.

July 16, 2017

Automatic Inventory Identification through Image Recognition

by
Matt Triff

Thesis submitted to Saint Mary's University
in partial fulfilment of the requirements for the
Degree of Master of Science in Computing and Data Analytics

July, 2017, Halifax, Nova Scotia

Approved:_____
Dr. Hai Wang
Supervisor
Department of Finance, Info Systems
and Management Science

Approved:_____
Dr. Haiyi Zhang
External Examiner
Acadia University

Approved:_____
Dr. Yasushi Akiyama
Supervisory Committee Member
Department of Mathematics
and Computing Science

Date: August 31, 2017

Extended Abstract

Companies are looking to use technology to replace repetitive, low skill tasks currently performed by human employees. One such series of tasks is conducting an inventory of items within a warehouse. This requires employees to identify individual items on a shelf. This task is well-suited to the application of computer vision techniques. Images of shelves in a warehouse can be acquired and analyzed. The goal of this research is to accurately identify items within the images for the purpose of inventory management.

This objective requires both the detection of objects within an image, and the identification of those objects. I propose a series of techniques to identify an item against a set of template images of potential items. The first step requires detection of the items in the image. The second step uses colour clustering and keypoint description to identify the item against a set of template images. An item in an image may be occluded and appear at various angles. The proposed techniques are designed to handle such distortions.

To demonstrate the effectiveness of these techniques, I have developed a prototype system that performs an inventory of equipment in a data centre. The prototype system provides a ranked list of possible matches for the pieces of equipment. The desired output is a limited set of possible matches that a human operator can easily verify. The prototype proved to be reasonably effective against 47 template images in ideal conditions. The ranked results provided the correct match within the top ten options 86% of the time, within the top five options 59% of the time, and as the first option 27% of the time. When tested against 74 template images and a more varied dataset the system was less effective, possibly due to imperfect variations in lighting and camera distortion. The ranked results provided the correct match within the top ten options only 12% of the time, within the top five options 10% of the time, and as the first option 6% of the time.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation and Application

In 2015, Saint Mary's University was approached to explore development of an image recognition system by an industry partner. The desired system would be used to boost productivity by reducing the labour involved in performing manual inventories in a warehouse environment. This lead to my involvement and the development of the proposed system described by this thesis.

The industry partner was unable to develop a large, labelled, dataset. Instead, they could only to provide a relatively small sample of images. Image recognition using machine learning has been popularized due to the quality of results it delivers [1]. However, machine learning algorithms require large datasets in order to prevent overfitting. This thesis describes the practical application of pattern recognition techniques to a limited dataset for image recognition. The desired system provides a list of potential matches to an input image. Ideally, the correct match would rank within the top ten options or better. The system would provide the ability for a low-skilled operator to quickly find the correct match from the results.

1.2 Thesis Scope

This thesis describes a potential approach to the generalized problem of performing an automated inventory of items within a warehouse. The proposed solution suggests an ensemble of pattern recognition techniques using edge detection, feature extraction and colour comparison. The use of these techniques individually is not novel. However, this thesis discusses some solutions to specific problems that have not been widely explored. One such solution is an algorithm to automatically "break down" an image into sub-images, such that specific parts of the image are removed and ignored, and the sub-images may be used with existing implementations of pattern recognition algorithms. Another solution is a process of deciding between two very similar images by analyzing specific areas of an image that are known to contain

identifying marks.

The proposed approach is applied to a data centre environment with the aim of identifying pieces of equipment in server racks. The dataset for this experiment consists of images taken of server racks within the data centre by two different types of cameras. This thesis shows where the proposed approach can best be applied for reasonable results. I also describe where the application of this approach will have poor results and the limitations of the proposed techniques.

In the experiments, the system has been developed in C# and C++, using the open source OpenCV computer vision library.

1.3 Thesis Structure

This thesis is divided into 6 chapters. Their contents are as follows:

Chapter 1: Introduction - Introduces the thesis and discusses the motivation and scope.

Chapter 2: Review of Literature - Provides an overview of the background knowledge required for this topic. Includes a review of relevant preprocessing, edge detection, and region matching techniques.

Chapter 3: Problem Statement - Provides an overview of the challenge to be solved by the proposed approach. Includes an overview and examples of the datasets being analyzed.

Chapter 4: Proposed Approach - Provides a theoretical overview of the proposed approach, including solutions to challenges and high level algorithms.

Chapter 5: Experimental Evaluation - Applies the proposed approach to a specific domain, and describes the algorithms and results of image recognition tests.

Chapter 6: Summary and Conclusion - Concludes the thesis and discusses the limitations and possible future directions for this research.

2 Review of Literature

2.1 Image Representation

OpenCV is an open source set of tools for computer vision tasks and algorithms. In OpenCV, images are represented as a two dimensional matrix of value elements. Each element in the matrix corresponds to a pixel in the image. Each element is a scalar containing values that describe the colour of the pixel. For example, a colour image matrix would contain elements with scalars containing three values for the red, green, and blue colour channels that make up the pixel. Similarly, for grayscale images, the scalar would contain only one value, between 0 and 255, where 0 is black and 255 is white.

A specific pixel within a 2D image matrix is specified using the x and y indexes. The image is organized in the matrix such that point (0, 0) is the top left corner of the image. An image can be cropped or extracted by copying the elements from the desired region to a new matrix matching the dimensions of this region.

Throughout this thesis, all algorithms referring to an image will be represented by OpenCV's representation [5].

2.2 Preprocessing

In this thesis, preprocessing refers to all tasks that prepare an image for the detection of objects. The following subsections describe the various literature and algorithms used in these tasks.

2.2.1 Calibration

Different cameras and lens take images differently. Specifically, cameras with a fish-eye lens have obvious distortion, as the edges are stretched and curved. This poses challenges when attempting to apply computer vision techniques to these images. In order to handle this issue, calibration can be used to convert images from their original, distorted form to a more accurate format.

There are many different calibration techniques available. The calibration techniques generally belong to one of two categories [49], photogrammetric and self-calibration. Photogrammetric calibration uses a single camera taking multiple pictures of a known 3D object. Self-calibration does not require a known 3D object. Instead, self-calibration uses a single camera moved through a static scene to calculate the required parameters. Zhang [49] proposed a commonly used approach that falls somewhere between the two categories [14]. Zhang's approach uses a 2D image, such as a checkerboard of known dimensions, instead of a 3D object. The checkerboard pattern is then mounted on a flat, planar surface. The user takes at least two (but often many more) images of the checkerboard in multiple locations within the frame of the image. These images, with a checkerboard of known size, are then processed to determine the required parameters for calibration. The checkerboard is detected in each image, and the size and shape is compared at the various locations within the image.

Zhang's method for calculating the calibration parameters is described as follows. The relationship between a 3D point captured in an image, M, and its location in the image projection, m, is shown by Equation (1), where s is an arbitrary scale factor, $\widetilde{m}$ is the 2D point vector, $[u, v]^T$, augmented with an additional element of 1, $[u, v, 1]^T$, A is the intrinsic parameters of the camera, defined by (2), R, t are the extrinsic parameters, rotation and translation, $\widetilde{M}$ is the 3D point vector, $[X, Y, Z]^T$, augmented with an additional element of 1, $[X, Y, Z]^T$. The calibration method determines the intrinsic parameters, as shown in Equation (2), the extrinsic parameters, as shown in Equation (1), and the coefficients of radial distortion, as shown in Equation (3).

Within Equation (2), the intrinsic parameters are defined as the following. (u_0, v_0) are the coordinates of the principal, or centre, point in the image, α and β are scale factors along the u and v axes respectively, and γ describes the skewness of the two axes of the image. These values uniquely define the attributes of a lens. Within Equation (3), (x, y) are the coordinates of an ideal, non-distorted point, and $(\breve{x}, \breve{y})$ are the coordinates of the real, distorted point. k_1 and k_2 are the coefficients of radial distortion. To determine all the required parameters for an image with radial

1. Detect the feature points of the checkerboard pattern in the images
2. Estimate the five intrinsic parameters and all the extrinsic parameters using Zhang's [49] closed-form solution
3. Estimate the coefficients of radial distortion by solving Zhang's linear least-squares equation
4. Refine all the parameters by minimizing Zhang's functions

Figure 1: Calibration procedure, as proposed by Zhang [49]

distortion (such as a fish eye lens), the process involves four steps, as shown in Figure 1. Equations (1) and (3) can then be applied to each pixel in the image to undistort the image.

$$s \cdot \widetilde{m} = A \cdot [R \quad t] \cdot \widetilde{M} \tag{1}$$

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\begin{aligned} \breve{x} &= x + x \cdot [k_1 \cdot (x^2 + y^2) + k_2 \cdot (x^2 + y^2)^2] \\ \breve{y} &= y + y \cdot [k_1 \cdot (x^2 + y^2) + k_2 \cdot (x^2 + y^2)^2] \end{aligned} \tag{3}$$

2.2.2 Thresholding

Image thresholding is the practise of separating objects in the foreground of an image from those in the background [46]. This task can be simple for images in which

the gray levels of the foreground vary significantly from those of the background. However, difficulties arise when attempting to segment images with issues such as significant noise, busyness of gray levels in an image, ambient illumination, and more [40]. There are a wide variety of thresholding algorithms available. The algorithms can largely be divided into six categories: histogram shape-based, clustering-based, entropy-based, object attribute-based, spatial, and local methods. Of the six categorizations, histogram shape-based thresholding is the most well known, and perhaps the most common [40]. Histogram shape-based techniques work by sorting the gray level values of the pixels within an image into a histogram. The peaks, valleys, and curvatures of the histogram are then analyzed to determine the optimal threshold value.

Otsu proposed a method to automatically select the threshold value based on the histogram of an image [31]. An image is first converted from colour to grayscale. This results in pixel values ranging from 0 (black) to 1 (white). The pixels are then grouped into histogram bins based on their value. Otsu proposed that well-thresholded classes would be separated in gray levels. Therefore, the threshold with the best separation of classes would also be the best threshold. To determine the optimal threshold value, we calculate either the within-class variance, as shown in Equation (4), or the between-class variance, as shown in Equation (5). Equation (6) shows the definition of the variables in Equations (4) and (5). The optimal threshold is such that either the within-class threshold is minimized, or the between-class variance is maximized.

$$\sigma_W^2 = \omega_b \cdot \sigma_b^2 + \omega_f \cdot \sigma_f^2 \tag{4}$$

$$\sigma_B^2 = \omega_b \cdot \omega_f \cdot (\mu_b - \mu_f)^2 \tag{5}$$

$$
\begin{aligned}
\omega_b &= \text{Weight of the foreground pixels} \\
\omega_f &= \text{Weight of the background pixels} \\
\mu_b &= \text{Mean pixel value of the background pixels} \\
\mu_f &= \text{Mean pixel value of the foreground pixels} \\
\sigma_b^2 &= \text{Variance of the pixels in the background} \\
\sigma_f^2 &= \text{Variance of the pixels in the foreground}
\end{aligned} \tag{6}
$$

The Otsu method has been found to be one of the better automatic methods for separating large objects from a background [40]. These images have a histogram of bimodal or multimodal distribution [30]. There have since been many modifications to the Otsu method to address specific alternative situations, such as unimodal distributions [30], segmenting small objects, [34] and segmenting objects under uneven lighting conditions [13].

2.3 Edge Detection

Within an image edges represent a significant local intensity change. In an image, these edges provide an appropriate method for the identification of different objects, or separate regions [2]. The following subsections describe the various techniques and algorithms related to the detection of edges.

2.3.1 Canny Edge Detection

The Canny edge detection algorithm, first proposed by John Canny in 1986 [7], is a popular edge detector. Edge detection consists of multiple stages; noise reduction, finding the intensity gradient, non-maximum suppression, and hysteresis thresholding, as shown in Figure 3.

The first stage is to use a five-by-five Gaussian filter to remove noise from the image. Noise has a negative effect on the rest of the algorithm, and may result in

spurious edges being detected. A Gaussian filter is a linear smoothing filter that is implemented through the use of a weighted sum of the pixels in a given window [16]. In this case, the window is five-by-five pixels. The weight of each pixel within the window is based upon the shape of the Gaussian function. Image processing uses a zero-mean discrete Gaussian function, shown by Equation (7). Since Gaussian filters are symmetric, rotation does not affect their output. Additionally, Gaussian filters are computationally efficient.

$$g[i,j] = e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{7}$$

In the second stage of the Canny edge detection algorithm, the image is filtered horizontally and vertically using a Sobel filter. Typically, a neighbourhood of three-by-three pixels is used. A Sobel operator computes the magnitude of a gradient, as shown in Equation (8) [17]. The partial derivatives of the magnitude can be computed by Equation (9). Equation (9) results in the masks shown in Equation (10). Since gradient direction is always perpendicular to edges, this step provides an initial set of edges.

$$M = \sqrt{s_x^2 + s_y^2} \tag{8}$$

$$\begin{aligned} s_x &= (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \\ s_y &= (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \end{aligned} \tag{9}$$

$$s_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad s_y = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \tag{10}$$

The third step in the Canny edge detection algorithm is non-maximum suppression. Non-maximum suppression step handles localization of an edge. That is, it aims to find the most accurate location of the edge. Only a one-pixel width line

should be returned by the algorithm for each individual edge. This step works by finding the pixel within an edge that has the greatest value. For this step, the horizontal and vertical orientation gradients generated by the second step are used. Each pixel on an edge is tested along the gradient. Since the gradient will not necessarily be the same angle as the grid of pixels in the image, linear interpolation is used to determine the values for comparison. Along the gradient, the values for the two nearest pixels are used to interpolate the value that is compared to the edge pixel. This is performed on both sides of the pixel being tested. The tested pixel is only kept as an edge if its value is greater than both adjacent, interpolated values. That is, for the pixel q in Figure 2, q must satisfy the requirements listed in Equation (11).

$$\begin{aligned} q &> r \\ q &> \alpha \cdot b + (1 - \alpha) \cdot a \\ &\text{and} \\ q &> p \end{aligned} \tag{11}$$

Finally, hysteresis thresholding is used to remove weak edges and to connect edges that have been split. The hysteresis step has two stages. The first uses two thresholds, T_{high} and T_{low}. Using the output from non-maximum suppression, this thresholding determines if a pixel is strong, weak, or a candidate. If a value is greater than T_{high}, it is a strong edge. If a value is less than T_{low}, it is a weak edge and is discarded. The values between T_{high} and T_{low} are called candidate pixels. Originally, T_{high} and T_{low} were determined empirically. However, further work has shown that using Otsu's method on the original image can determine T_{high}, and half that value has been generally found to be a good T_{low} value [9]. Candidate pixels are deemed to be part of an edge if they are connected to a strong pixel within a their local neighbourhood.

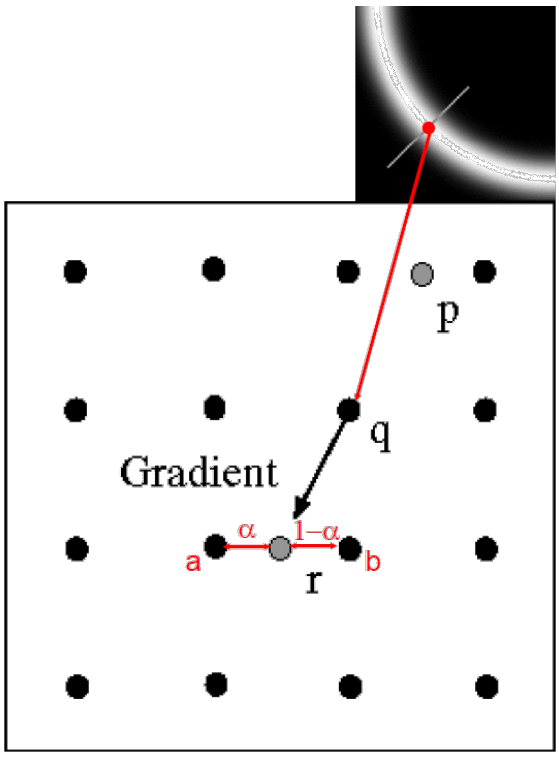

Figure 2: Depiction of linear interpolation, used for non-maximum suppression by the Canny edge detector [47]

1. Smooth the image with a Gaussian filter.
2. Compute the gradient magnitude and orientation using a Sobel operator.
3. Apply non-maximum suppression to the gradient magnitude.
4. Use the hysteresis thresholding algorithm to detect and link edges.

Figure 3: Steps for the Canny Edge detection algorithm

2.3.2 Progressive Probabilistic Hough Transform

The Hough transform provides a voting method for detecting imperfect matches to a line in complex patterns of points (pixels) in an image. The original Hough transform was developed for straight lines. It has since been expanded to detect any number of complex shapes [3]. For this thesis' purposes, only the straight line detection is required.

Provided a set of edges, such as those provided by Canny edge detection, the Hough transform can detect patterns even when the match is imperfect. To detect a straight line, the Hough transform works by defining a straight line by Equation (12),

$$r = x \cdot \cos\theta + y \cdot \sin\theta \tag{12}$$

where r is the distance from the origin to the closest point on the line,
θ is defined as the angle between the x-axis and the line of length r.

For each point within the set of edges, lines are plotted going through the point at all angles from 0 to 360 degrees. With each plotted line, r and θ are measured and can be plotted. At every point where the plotted lines intersect, we know that the same line passes through the two points specified by those lines, as shown in Figure 4. All collisions where the same line passes through multiple points are accumulated. By using a user-specified threshold value, we can determine how many points are required to collide before a line is determined to exist within the image. The threshold may be determined ad-hoc, or systematically, such as by using one-half of the largest number of points.

The Probabilistic Hough Transform (PHT), proposed by Kiryati et. al. [20], performs the original Hough Transform algorithm on a preselected fraction of input points. This method often achieves results similar or identical to those of the original Hough transform algorithm with increased performance.

Progressive Probabilistic Hough Transform [25] builds upon PHT by further minimizing computational requirements. The algorithm is best described in the outline provided by Matas et al., shown in Algorithm 1. The creators of the algorithm deter-

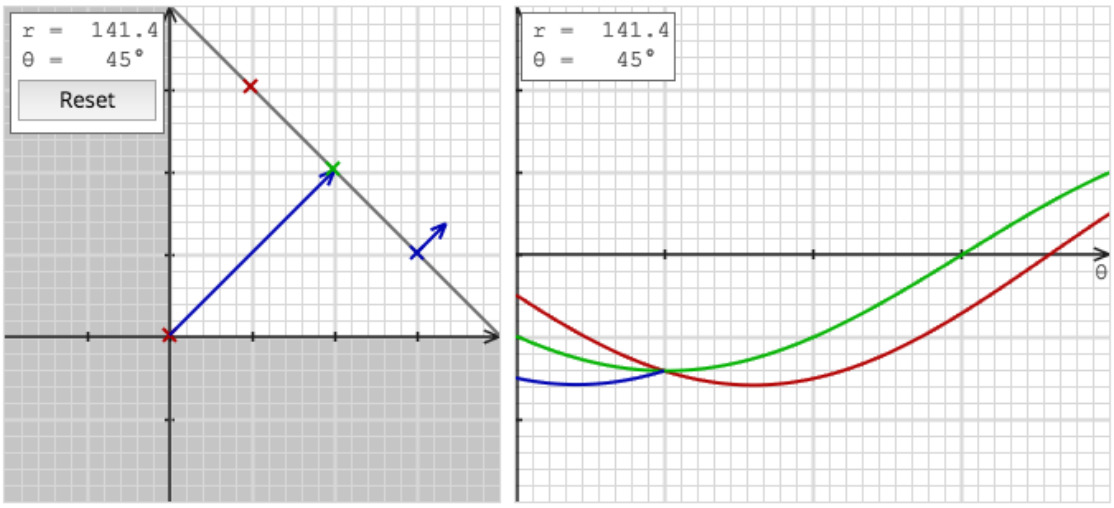

Figure 4: The graphs show three points. On the left is the line drawn between the three points. On the right are the results of plotting the points by Equation (12)

mined via experimental tests of the algorithm that it results in similar line detection output, with significantly fewer operations.

2.4 Region Matching

This study uses two measures to match regions to one another. The first is image features, the second is colour. The methods to determine the key features and colours in an image are built up of many other techniques. The two techniques used in this study is Oriented FAST and Rotated Brief (ORB) for feature matching, and colour clustering for colour matching. The following subsections describe the various techniques and algorithms used by these methods to detect and define the key features and colours in each image.

2.4.1 Features from Accelerated Segment Test (FAST)

In computer vision, features are key areas of an image that can be used to uniquely describe it. Typically, they are points of interest on the image where the foreground

Algorithm 1 Progressive Probabilistic Hough Transform algorithm outline [25]

```
1: while image not empty do
2:    Randomly select a pixel from the image
3:    Update accumulator with the pixel
4:    Remove the pixel from input image
5:    if the highest peak in the accumulator is higher than threshold thr(N) then
6:       Look along a corridor specified by the peak in the accumulator
7:       Find the longest segment that is continuous, or exhibits a gap not exceeding
         a given threshold
8:       Remove the pixels in the segment from the input image
9:       "Unvote" from the accumulator all the pixels from the line that have previ-
         ously voted
10:      if the line segment is longer than the minimum length then
11:         Add the line segment to the output list
12:      end if
13:   end if
14: end while
```

meets the background or there is some other change of colour.

Features from Accelerated Segment Test (FAST) is a feature detector that uses corner detection and emphasizes performance [37]. FAST was originally developed to handle feature detection in live video at full frame rate. This task requires fast processing performance. Both the speed and capability of the algorithm has resulted in widespread use in other computer vision tasks as well.

The FAST algorithm consists of three stages: segmentation test, machine learning and non-maximal suppression. The segmentation test used by FAST is based on previous feature detectors. When deciding whether a pixel, p, is a corner, the segment test examines a circle of sixteen pixels that surround p. If n contiguous pixels within the circle are either all darker, or all lighter than p, plus or minus some threshold t, it is considered a feature. With n equal to twelve, the test only needs to examine four pixels to reach a negative conclusion. At least three of the surrounding pixels need to be lighter or darker than p. If this condition is satisfied, the rest of the pixels can be checked; otherwise, the pixel can be discarded. However, this approach has

four weaknesses:

1. It does not generalize well for n less than twelve
2. The choice and order of the fast test pixels contain assumptions about the feature
3. The knowledge from the first four tests is discarded
4. Multiple features are often detected adjacent to one another (when it should be defined as one feature)

FAST overcomes the first three obstacles by using machine learning to develop a decision tree. The decision tree is trained preferably on a set of images from the application domain, although this is not strictly necessary. The segment test is performed on the training images and p, and the surrounding sixteen pixels are stored in a vector. The vectors are stored and partitioned into three sets: the lighter set, darker set and the similar set. Using the Iterative Dichotomiser 3 method, a decision tree is developed based on recursively testing for the surrounding pixel that most impacts the decision on whether p is a corner. The decision tree is then turned into nested if-then-else statements, which can be used to test for corners. Although the results are not exactly the same, the decision tree mimics the results from the segmentation test. Further optimization may be performed to remove redundant sub-trees within the decision tree.

Non-maximal suppression is used to fix the issue of multiple features being detected adjacent to one another. Adjacent keypoints are compared using the function V, as shown in Equation (13), which is the sum of the absolute difference between p and its sixteen surrounding pixels. FAST can be performed with or without non-maximal suppression.

$$V = max\left(\sum_{x \in S_{bright}} |I_{p \to x} - I_p| - t, \sum_{x \in S_{dark}} |I_p - I_{p \to x}| - t\right) \tag{13}$$

2.4.2 Binary Robust Independent Elementary Features (BRIEF)

Binary Robust Independent Elementary Features (BRIEF) is a fast feature detector [6]. BRIEF works by comparing binary strings that describe feature points within an image. This has distinct memory advantages over previously invented methods, which used large-dimension vectors of floating point numbers. Furthermore, binary strings can be compared using the very efficient Hamming distance calculation. BRIEF is not rotation invariant, although it performs reasonably well on images with some rotation.

BRIEF first smooths a square patch of dimension S on the image to remove and reduce the effect of noise on the descriptor. The patch is then sampled using the Gaussian equation in Equation (14), which was experimentally determined to provide the best results [6]. This equation matches the pattern shown by Figure 5. The process is repeated for n patches throughout the image to create a large set of descriptors. Hamming distance between the binary descriptors is then used to determine the closest matches.

$$(X, Y) \sim \text{i.i.d. } Gaussian(0, \frac{1}{25}S^2) \tag{14}$$

where (X, Y) is the point being sampled,
S is the width of the patch.

2.4.3 Oriented FAST and Rotated BRIEF (ORB)

Oriented FAST and Rotated BRIEF (ORB) is a feature detector that builds upon the FAST and BRIEF algorithms [39]. ORB adds three key improvements over the original algorithms. The first is the addition of orientation to FAST. Orientation of the corners detected by FAST is based on the intensity centroid method proposed by Rosin [36].

The second improvement is the efficient computation of oriented BRIEF fea-

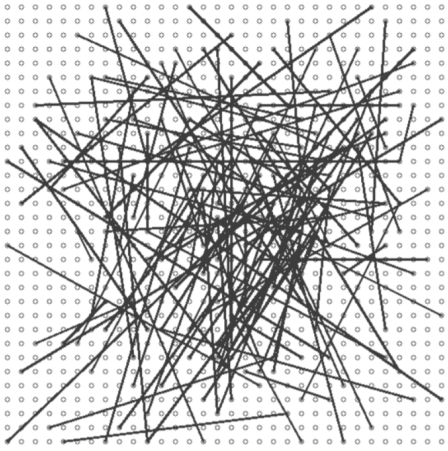

Figure 5: Test location selection method using Equation (14) for BRIEF [6]

tures. To avoid high computation when orienting BRIEF, ORB instead proposes pre-computing a lookup table for BRIEF bit string patterns. The orientation of the keypoints are used to determine the rotation matrix for the image. Using the rotation matrix, the keypoints are rotated, or steered, to match the values in the look up table.

Finally, the algorithm proposes a learning method for BRIEF to improve nearest neighbour applications by de-correlating BRIEF features under rotational invariance. To train the learning method, ORB uses a greedy search of all the potential binary tests of pixel patches on a training image dataset. The greedy search is performed until the 256 most uncorrelated tests are determined, with a mean result around 0.5. This set of binary tests can then be used to efficiently test input images for descriptors.

2.4.4 k-Nearest Neighbour

The k-Nearest Neighbour algorithm is a fundamental, straightforward algorithm used for classification [32]. First, the algorithm uses a set of labelled training vectors of size N, of multiple classes. Then, given the vector you wish to classify, c, the algorithm computes the k nearest vectors to the vector c, where nearest is defined by some

function to measure distance or similarity. Of the k nearest neighbours, the class that makes up the majority of the neighbours is used to classify the class of the vector c. To prevent ties, the value k should be an odd number that is not a multiple of the number of classes.

2.4.5 Least Median of Squares (LMedS)

Least Median of Squares (LMedS) is a robust regression method, used to find the relationship between multiple variables. LMedS builds upon Least Squares regression. A linear model of a set of points is classically defined by Equation (15).

$$y_i = x_{i1} \cdot \theta_1 + ... + x_{ip} \cdot \theta_p + \sum_{i=1}^{n} e_i \tag{15}$$

where e_i is the error, normally distributed with a mean of zero.

Regression methods aim to estimate $\theta = (\theta_1, ..., \theta_p)^t$ based on the data $(x_{i1}, ..., x_{ip}, y_i)$, which is defined as $\hat{\theta}$. Least squares, was originally defined by Gauss or Legendre (under dispute, [43]) is defined by Equation (16).

$$\underset{\hat{\theta}}{minimize} \quad \sum_{i=1}^{n} r_i^2 \tag{16}$$

where r_i^2 is the right-hand side of Equation (15), without the error.

Least squares seeks to minimize the sum of squared residuals for a set of points. Rousseeuw proposes the usage of the median of squared residuals instead as the parameter to minimize, as shown by Equation (17) [38].

$$\underset{\hat{\theta}}{minimize} \quad \underset{i}{med} \quad r_i^2 \tag{17}$$

This method is shown to be resistant to contamination of data, as it resists outliers and false matches. This is especially valuable for computer vision, where noise can

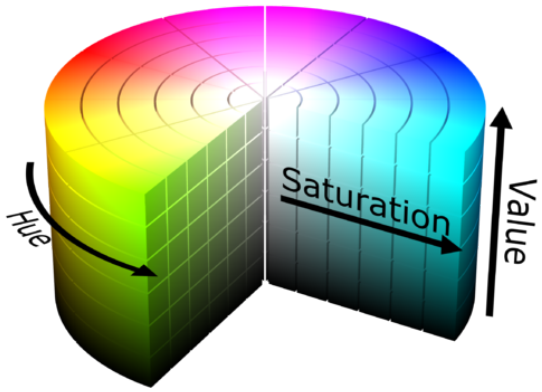

Figure 6: HSV colour model mapped to a cylinder [41]

have significant effects and contamination.

2.4.6 Hue, Saturation, Value (HSV) Colour Space

Although colour in computer displays is typically modelled in the RGB (red, green, blue) colour space, there exist many other colour spaces for modelling a specific colour. This study makes use of the HSV (hue, saturation, value) colour space [42].

Hue is an attribute of colour based on its dominant wavelength. Hue distinguishes between different colours such as red, green, blue, etc. Saturation of a colour is the colourfulness compared to its lightness [18]. A colour with high saturation is perceived as more colourful. A colour with low saturation is perceived to be "washed-out". Similarly the final dimension, value, is the colourfulness compared to the colour's darkness. A colour with low value is entirely black, whereas a colour with high value has no black. HSV can be modelled as a cylinder, as shown in Figure 6.

An RGB pixel can be converted to the HSV colour space using the algorithm provided in Algorithm 2.

Algorithm 2 Algorithm to convert a pixel from the RGB to HSV colour space, from [42]

$V := \max(R, G, B)$
Let $X := \min(R, G, B)$
$S := \frac{V-X}{V}$, if $S = 0$ return
Let $r = \frac{V-R}{V-X}, g = \frac{V-G}{V-X}, b = \frac{V-B}{V-X}$
if $R = V$ **then**
    if $G = X$ **then**
        $H = 5 + b$
    else
        $H = 1$ - g
    end if
else if $G = V$ **then**
    if $B = X$ **then**
        $H = 1 + r$
    else
        $H = 3$ - b
    end if
else
    if $R = X$ **then**
        $H = 3 + g$
    else
        $H = 5$ - r
    end if
end if
$H = \frac{H}{6}$

2.4.7 Colour Indexing

Colour indexing is a common method for providing a succinct representation of an image [44]. An image is converted from its 2D representation into a histogram of N specified bins along each of its channels. Each channel range is broken into N equal bins. For each pixel, the pixel value in each channel is sorted into the appropriate bin. For a typical RGB or HSV image, a colour index of the image would consist of a total of $3N$ values. This provides a fixed-length representation no matter the shape or size of the image being analyzed.

The fixed-length representation makes colour indexing using histograms an effective technique for comparing different images. Once the histograms have been created, they can be normalized based on their own values. The histograms can then be compared through a number of methods, including correlation, Chi-Squares, intersection, and Bhattacharyya distance. Correlation, which was used in this study, is shown in Equation (18).

$$d(H_1, H_2) = \frac{\sum_{i=1}^{D}(H_{1i} - \bar{H_1})(H_{2i} - \bar{H_2})}{\sqrt{\sum_{i=1}^{D}(H_{1i} - \bar{H_1})^2 \sum_{i=1}^{D}(H_{2i} - \bar{H_2})^2}} \tag{18}$$

where H_1 and H_2 are the two histograms being compared,
i is a dimension of the histogram,
D is the total number of dimensions in the histogram,
$\bar{H_k}$ is defined by Equation (19)

$$\bar{H_k} = \frac{1}{N} \cdot \sum_{j=1}^{N} H_{kj} \tag{19}$$

where N is the total number of bins in the histogram.

Colour indexing has the added benefit of being invariant to rotation [44]. An image will always have the same distribution of colours when analyzed from any orientation. The occlusion of an object in one of the images being compared will

result in a change in the histogram in proportion to the amount of the object that is occluded. Similarly, the distance of the object will proportionally affect the histogram the further away the object is.

2.4.8 K-means Clustering

k-means is a popular statistical clustering method [22, 24]. The objective of the algorithm is to take a finite set of m-dimensional vectors, and group them into k clusters, where k is specified by the operator when running the algorithm. Each cluster is defined by its centroid, the centre point of the cluster. k-means initially picks k number of random vectors to serve as the centroids of the clusters. The algorithm iteratively runs by assigning each vector based on the minimum distance between the vector and a cluster centroid. After all vectors have been assigned to a cluster, the centroids are recomputed, as shown in Equation (20). The process repeats until either n iterations have completed, or the change in location of the centroids between iterations falls below a threshold t.

$$\vec{c}_i = \frac{\sum_{\vec{x} \in \vec{c}_i} \vec{x}}{\mid \vec{C}_i \mid} \tag{20}$$

where $1 \leq k$,
$\vec{x}$ is an object vector,
$\mid \vec{C}_i \mid$ is the cardinality of the the cluster $\vec{C}_i$.

To determine the quality of the clusters, a number of different methods have been proposed, as summarized by Halkidi et al. [11]. Common validity measures rely on within-cluster scatter or between-cluster separation. Within-cluster scatter can be computed for each cluster by Equation (21),

$$S_i = \frac{1}{\mid \vec{C}_i \mid} \sum_{\vec{x} \in \vec{c}_i} distance(\vec{x}, \vec{c}_i) \tag{21}$$

where S_i is the scatter within the ith cluster.

Within-cluster scatter for the set of clusters is then defined as the sum of the scatter for each cluster, as shown in Equation (22).

$$S(C) = \sum_{i=1}^{k} S_i \tag{22}$$

Between-cluster separation can be computed for a cluster by comparing the distance between different clusters, as shown in Equation (23).

$$d_{ij} = distance(\vec{c}_i, \vec{c}_j) \tag{23}$$

Between-cluster separation for the set of clusters is the sum of the distance between all clusters and each other cluster, as in Equation (24).

$$D(C) = \sum_{i=1}^{k} \sum_{j=1}^{k} d_{ij} \tag{24}$$

Because the original centroids are picked at random, it is recommended that the algorithm be run a number of times in order to avoid potential local maximizations. The result that provides the best clustering (as described by a validity measure) is kept.

When choosing the number of clusters to use, it is recommended that the dataset be tested with a large number of cluster sizes. The quality, as defined by the validity measure, can then be plotted. The knee of the curves can be used to determine the number of appropriate clusters. For example, typically the within-cluster scatter will rise rapidly when the value of k falls below a certain size. Similarly, the between-cluster separation will fall rapidly when the value of k falls below a certain size.

2.4.9 Colour Clustering

Colour clustering is a common technique used for image segmentation [23]. By applying clustering algorithms, such as k-means, to the pixels of an image, objects within the image can be identified. This provided the ability to distinctly identify objects in the foreground and background of an image.

Kankanelli et. al. [19] proposed a colour clustering technique for matching images. Their proposed method uses the colour indexing technique, as proposed by Swain and Ballard [44], to find the centroids of their clusters. In their approach, each peak in the colour indexing histogram represents a centroid. To compare two images via colour clustering, a distance measure can be used to compare the results of the two clusters. This comparison can use a simple distance measure, such as Euclidean distance, or a more complex weighted distance measure, based on the frequency of the centroid. Kankanelli et. al.'s [19] preliminary results suggested greatly improved performance compared to traditional colour indexing techniques via histogram alone.

3 Problem Statement

3.1 Overview

The application domain for the techniques proposed by this research is a data centre. Data centres contain many racks of equipment used for information processing and storage. Computer server racks are mainly found in data centres. This study will show how the techniques proposed by this research can provide automatic detection and identification of the equipment within the server racks. To demonstrate the feasibility of the proposed techniques, a prototype system has been built. The prototype system is further described in Chapter 4.

The data used in this study consists of images and data models in JavaScript Object Notation (JSON) format. The first set of images are of the server racks found in the data centre. These server racks contain many pieces of equipment. This is the set of images that will be analyzed by the system on an ongoing basis. The second set of images are classified as template images. These are the templates that are used to identify a given piece of equipment. Each piece of equipment that can be identified in a server rack image has at least one corresponding template image. Some equipment have two corresponding template images, for when a piece of equipment may or may not have a cover. All images used in this study were provided by the industry partner.

3.2 Server Rack Images

For the purposes of this study, server rack images were taken with two different cameras. A small sample was initially taken with a Canon EOS 20D camera. Due to the limited field of view, only a small part of the server rack was captured in one photo. Three photos were required to capture an entire server rack. The three photos were manually stitched together to create a single image, as shown in Figure 11.

A larger dataset was created using a GoPro Hero 4 camera. The GoPro Hero 4 is

a common action camera, typically used by adventurists. For our purposes, the most beneficial aspect of the camera is that it has a fish-eye lens, which provides a wide field of view. The wide field of view allows for a single image to capture an entire server rack. However, this does result in some distortion that needs to be corrected. Additionally, due to the wide field of view, the server rack images sometimes contain a portion of the adjacent server racks to the left and right of the server rack to be analyzed. The images occasionally also contain glare from overhead lighting, partially obscuring the equipment, the server rack frame, or both. The prototype system, as detailed in Chapter 4, makes the required corrections

Server racks are measured in a unit of measurement called a "rack unit". One rack unit is equivalent to 44.45 millimetres or 1.75 inches in height, and can be written in the notation 1RU. Two rack units is written as 2RU, three rack units as 3RU, etc. Server racks can range in height anywhere between 9RU and 49RU. For the purposes of this study, the server racks analyzed were either 16RU or 49RU. The width of a server rack can also vary, with standard sizes of 19 inches or 23 inches. Equipment that is 19 inches in width may fit into server racks that are 23 inches in width with special adaptors. Equipment that is 23 inches in width cannot fit in 19 inch wide racks.

3.2.1 Data Model

The data model for a server rack consists of high level attributes that pertain to the entire server rack and an array of server objects that describe the equipment within the rack. The completed system outputs the data model in JSON format after performing the analysis of an image. A skeleton of the data model is shown in Figure 7. The top level attributes describe the physical characteristics of the server rack. The characteristics are provided as initial input to the system. These attributes are: a unique identifying name, the height in rack units, and the width in inches. The server rack images are labelled, using their file name, in a format that is understandable by the system:

```
{
  "analyzedAt": "#/#/#### ##:##:## AM",
  "name": "RACK NAME",
  "height": ##,
  "width": ##,
  "servers": [
    {
      "ImagePath": "PATH/TO/IMAGE.JPG",
      "LocationRU": ##,
      "Server": [
        [
          {
            "Score": #####,
            "Value": "ImageLibraryDir/TEMPLATE IMAGE 1.jpg"
          },
          ...
          {
            "Score": #####,
            "Value": "ImageLibraryDir/TEMPLATE IMAGE X.jpg"
          }
        ]
      ]
    },
    ...
        {
      "ImagePath": "PATH/TO/IMAGE.JPG",
      "LocationRU": ##,
      "Server": [
        [
          {
            "Score": #####,
            "Value": "ImageLibraryDir/TEMPLATE IMAGE 1.jpg"
          },
          ...
          {
            "Score": #####,
            "Value": "ImageLibraryDir/TEMPLATE IMAGE X.jpg"
          }
        ]
      ]
    }
  ]
}
```

Figure 7: JSON format/skeleton for a server rack

$Name\_HHRU\_WWin.jpg$

where H is a numeric value corresponding to height,
WW is a numeric values corresponding to width.

In future iterations of the system these attributes could automatically be re-assigned based on the exact location where the image was taken, or based on the order the images of an entire data centre were taken.

A series of server objects are included in an array and correspond to each of the pieces of equipment in the rack that the server has identified. Each server object contains two attributes, "LocationRU" and "Server". LocationRU is the location within the server rack. For example, if the value is 30, then the server rack is 30RUs from the bottom of the rack. The server attribute is an identifier corresponding to a template image. In our case the identifier used was the image name.

3.2.2 Examples

Figure 8 shows an example of a raw image of a server rack. Figure 9 shows an example of an input server rack image. The corresponding data model output is shown in Figure 10. Figure 11 shows an example of glare from overhead lighting.

3.3 Template Images

Template images are used by the system to identify the pieces of equipment within the server racks. A single template is an image of an individual piece of equipment. These images server as a "definition" of what a piece of equipment looks like. Ideally, these images show the piece of equipment when it is first put into a server rack, before any cables are connected and before the piece of equipment is turned on. In ideal conditions the images would also be taken under similar lighting conditions, distance, and at the same angle as images of the server rack in normal operations. However, due to operational requirements these images could not be taken for this study. The

Figure 8: Raw image of a server rack taken with a GoPro 4 camera

Figure 9: Image of a server rack after undistortion and cropping

```
{
  "analyzedAt": "2/8/2016 11:07:05 AM",
  "name": "Rack 0012",
  "height": 49,
  "width": 19,
  "servers": [
    {
      "ImagePath": "temp/extractedServer_0_Rack 0012_49RU_19in_x1150y1159_x21710y2223.jpg",
      "LocationRU": 47.67,
      "Server": [
        [
          {
            "Score": 0.246316,
            "Value": "ImageLibraryDir/Telect GMT 10-10 2_1RU_19in.jpg"
          },
          {
            "Score": 0.336707097935103,
            "Value": "ImageLibraryDir/Drake VM2410A Modulator_1RU_19in.JPG"
          },
          ...
          {
            "Score": 0.87362395258151,
            "Value": "ImageLibraryDir/Telect KLM-GMT 4-4 Fuse Panel_1RU_19in.png"
          }
        ]
      ]
    },
    ...
    ...
    {
      "ImagePath": "temp/extractedServer_6354_Rack 0012_49RU_19in_x1150y16513_x21710y26978.jpg",
      "LocationRU": 5.0,
      "Server": [
        [
          {
            "Score": 0.380193333333333,
            "Value": "ImageLibraryDir/Telect GMT 10-10 2_1RU_19in.jpg"
          },
          {
            "Score": 0.383350508926857,
            "Value": "ImageLibraryDir/Harmonic NSG 9000 Eqam_2RU_19in.JPG"
          },
          ...
          {
            "Score": 0.75781,
            "Value": "ImageLibraryDir/Leitch 6800_2RU_19in.png"
          }
        ],
        [
          {
            "Score": 0.186424,
            "Value": "ImageLibraryDir/Telect GMT 10-10 2_1RU_19in.jpg"
          },
          {
            "Score": 0.431782766475605,
            "Value": "ImageLibraryDir/Harmonic NSG 9000 Eqam_2RU_19in.JPG"
          },
          ...
          {
            "Score": 0.718420666666667,
            "Value": "ImageLibraryDir/Leitch 6800_2RU_19in.png"
          }
        ]
      ]
    }
  ]
}
```


Figure 10: JSON output after analyzing a server rack image

Figure 11: Image of a server rack with glare from overhead lighting. Image is from preliminary tests, using a camera with a narrow field of view. This image is a composite of three separate images.

Figure 12: Image of a Dell PowerEdge R720xd piece of equipment

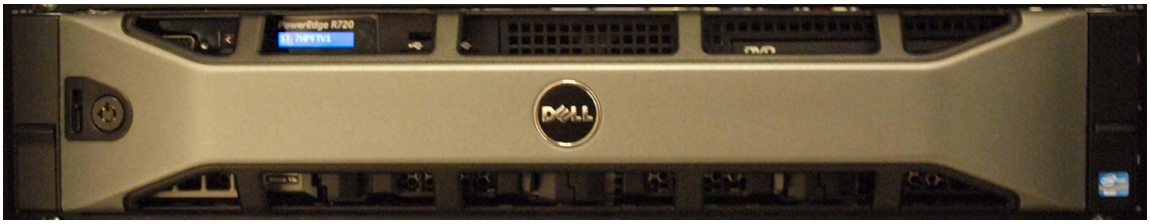

Figure 13: Image of a Dell PowerEdge R720 piece of equipment

template images were gathered from a variety of sources, either of the equipment in operation in a data centre, or from online sources such as the manufacturers website. This results in some template images where colour, lighting, and occlusions such as cables differ from a "clean" template image. The system was built to expect and withstand these conditions.

In other cases, a template image was extracted directly from the set of rack images. These templates often contain some occlusions. Where possible, these occlusions are ignored by the proposed approach.

There are two notable pieces of information that must be provided for the template images that are not obvious physical characteristics. The first are "features". These are rectangular areas on the image that the system should pay special attention to. For example, a model number, a manufacturer logo, or other label. These are important because although they may be small, they are critically important to identifying different models of a piece of equipment that may differ only very slightly from one another. For example, see Figure 12 and Figure 13 of two similar pieces of Dell equipment.

The second notable piece of information is the rectangular areas of the image

```
{
  "imagePath": "IMAGE NAME.jpg",
  "manufacturer": "MANUFACTURER",
  "model": "MODEL ####",
  "RU": #,
  "width": ##,
  "features": [
    {
      "x" : ###,
      "y" : ###,
      "height" : ###,
      "width" : ###,
      "weight": 0.#
    },
    ...
    {
      "x" : ###,
      "y" : ###,
      "height" : ###,
      "width" : ###,
      "weight": 0.#
    }
  ],
  "ignorableRegions": [
    {
      "x" : ###,
      "y" : ###,
      "height" : ###,
      "width" : ###
    },
    ...
    {
      "x" : ###,
      "y" : ###,
      "height" : ###,
      "width" : ###
    }
  ]
}
```

Figure 14: JSON format/skeleton for a server rack

deemed to be "ignorable regions". These are areas of the image that may differ slightly or significantly between different pieces of the same equipment. For example, a port may or may not have a cable plugged into it, or a light may or may not be on at the time a picture is taken.

3.3.1 Data Model

The data model for the template images contains physical information about the piece of equipment and information that will aid the system in identifying the pieces of equipment in the server racks. The empty data model is shown in Figure 14.

The attribute "imagePath" defines the path within the system's internal directory structure where the template image is stored. Similarly to the server rack data model, the attributes for manufacturer, model, RU, and width are defined in the filename,

using the format:

$$ManufacturerModel\_HRU\_WWin.jpg$$

where H is a numeric value corresponding to height,
WW is a numeric values corresponding to width.

Features, as described above, contains an array of objects that describe rectangular areas within the image. Feature objects have five values. The "x" and "y" values are the pixel location, within the image, of the upper left corner of the rectangle, where the origin of the coordinates is the top left of the image. The "width" and "height" values are the width and height of the rectangle, in pixels. The last attribute is "weight", which is used by the system to determine how important a given feature is compared to the other features. Ignorable regions is also an array, containing objects in the same format as feature objects, but without the weight attribute.

The data model filled out by an operator with knowledge of the piece of equipment. For this study, the values for the features and ignorable regions were determined using a simple web-based tool.

3.3.2 Examples

Figure 12 shows an example of a template image obtained from the web. An example of the data model for a templateis shown in Figure 15.

Figure 16 shows an example of a template image obtained from an operational environment.

Figure 17 shows the image with drawings of the features (outlined in green) and ignorable regions (filled in black) for 12.

```
{
  "imagePath": "Apex 1000_1RU_19in.jpg",
  "manufacturer": "Apex",
  "model": "1000",
  "RU": 1,
  "width": 19,
  "features": [
    {
      "x": 18,
      "y": 114,
      "height": 22,
      "width": 71,
      "weight": 0.8
    },{
      "x": 14,
      "y": 8,
      "height": 31,
      "width": 141,
      "weight": 0.6
    }
  ],
  "ignorableRegions": [
    {
      "x": 12,
      "y": 40,
      "height": 66,
      "width": 470
    },{
      "x": 1228,
      "y": 26,
      "height": 72,
      "width": 191
    },{
      "x": 485,
      "y": 94,
      "height": 25,
      "width": 150
    }
  ]
}
```


Figure 15: JSON format/skeleton for a piece of equipment template

Figure 16: Image of a Sun Microsystems SunFire V60x piece of equipment

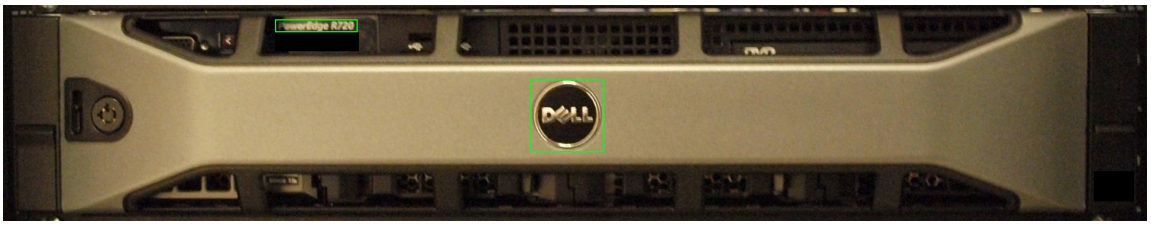

Figure 17: Dell PowerEdge R720, as shown in Figure 12, with the features outlined in green, ignorable regions filled in black

4 Proposed Approach

There have been large advances in computer vision in recent times. There exist a plethora of computer vision techniques that can be applied in various ways and in many combinations. However, the application of these techniques remain very domain and application specific.

Machine learning-related computer vision techniques are a potentially more general exception. However, even machine learning techniques require a large amount of training data in their required domain. Gathering and labelling large sets of training data is not always feasible or reasonable for every application.

Below, I propose a series of techniques to complete the two major tasks within the prototype system. They consist of applying existing algorithms, as well as some novel techniques specifically developed for the problem at hand. The two major tasks within the prototype system are matching a region to a known template, and isolating the region in order to analyze it. The latter I have termed "Edge Detection" and can be thought of as a preprocessing step for the second task, which I have termed "Region Matching".

4.1 Edge Detection

4.1.1 Objective

At the start of the process, the solution will receive as input the image of a warehouse shelf. The objective of this stage of the process is to isolate the individual items on the shelf.

4.1.2 Proposed Method

The act of isolating an object within an image will be very specific to the application domain. Items on a warehouse shelf will require different techniques than isolating a piece of a equipment in a server rack, or isolating a car in a parking garage. However, the same general stages can be applied to all cases. I discuss the exact

implementation used for locating pieces of equipment in a server rack in the next chapter. In the following discussion I will use the example of a warehouse shelf to describe the prototype system. Although as discussed before, the same techniques can be applied to other domains.

Given the field of view on a typical camera, it is expected that an image of a specific shelf will necessarily contain partial adjacent shelves. The first step is therefore to isolate the vertical shelves using the following novel technique. In a typical warehouse the shelves will all be of a standardized (or at the very least, similar) size and colour. We can use this knowledge to identify the vertical columns separating shelves.

First, we will convert the image of the shelves to black and white, using a reasonable threshold, such as that provided by Otsu's method. Next, we can search the image for a continuous region of pixels of the same value, while allowing for noise. Assumptions can be made about the left and right vertical columns appearing in the left and right quarters of the image, respectively. The same technique can be applied to locate the top-most, and bottom-most shelves. The image can then be cropped, based on the locations derived from the above algorithm.

Next, we must identify the individual products on the shelves. A standard Canny edge detector can be used to locate the edges of the objects in the image. We then aim to identify the horizontal lines that delineate the bottom and top of the box containing the product. If our shelves contain multiple products on each shelf, we would similarly look for vertical lines as well. The Hough transform is a popular and accurate technique for identifying such straight lines. The Hough transform will detect many straight lines. These lines are usually very short in length, around 10-20 pixels. Of all the straight lines identified by the Hough transform, we can ignore those lines that have an angle of plus or minus five degrees. We allow a variation of 10 degrees to allow for factors such as tilt in the camera at the time the image was taken, or uneven floors. The straight lines with a greater angle are likely images on the boxes, rather than the outlines of the boxes themselves.

After removing lines with a large angle, the remaining lines will be all of the straight, level, lines in the image. However, there will still be some straight lines

from the images on the product boxes. These spurious lines will be eliminated by keeping only the lines that are the most dominant across the entire image. This line elimination is accomplished by calculating the maximum number of lines that intersect with a given row of pixels. Any row of pixels that has less than a fixed percentage of the maximum number of lines should be discarded. Through experimentation, it was determined that 30% was a reasonable fixed percentage for this filtering.

Depending on domain knowledge, if you know that the objects you are looking for have either a fixed, or a minimum height, you can further eliminate some of the remaining lines. This can be accomplished by iterating through the image from top to bottom with a window of height equal to the minimum height of the objects you are identifying. Within that window you can remove all but the most dominant line.

This will result in the a reasonable approximation of the locations of the individual products within the image. A high level description of the algorithm is provided in Algorithm 5.

4.2 Region Matching

4.2.1 Objective

At this stage of the process, the solution will attempt to determine a ranked list of the best possible matches to the individual items that were isolated in the Edge Detection phase.

4.2.2 Proposed Method

As mentioned, the specified edge detection techniques will need to be customized to a specific application domain. However, the following region matching approach can be applied more generally to any set of images with the same metadata.

This technique is focused on providing a similarity score between a template image, and an image of another object, with the same rotation, and the taken from the same, or similar, angle. For the proposed algorithm, it is necessary to have

metadata about the template image. The metadata should specify the regions on the template image that are not expected to be the same, and are thus ignorable. For an item on a warehouse shelf, this could be something such as expiry date stickers, labels created during production or for special promotions. The metadata should also specify the regions on the template image that are the most important. These regions are known as features. For example, the label showing the model number, the label of the manufacturer, etc. The feature metadata should also specify a weight for each of the features. For example, the model number will be more important, and thus have a higher weight, than the feature showing the manufacturer.

These two sets of regions are important for different reasons. The ignorable regions help improve the image matching by not allowing these areas to affect the score when it is known ahead of time that they will be different. The feature regions help improve the matching by ensuring that specific attention is paid to the key areas that will differentiate between images that may otherwise be extremely similar to one another.

Given the set of ignorable regions, it is necessary to remove those regions from any future matching activities. Existing implementations of image matching algorithms do not provide for the possibility of ignorable regions [14]. To leverage previous work, and avoid the task of altering existing implementations of image matching algorithms, we can instead alter the image being matched. Ignorable regions may be located anywhere within an image, and have any potential rectangular shape. Removing the ignorable regions by setting the pixels to all black, or all white, or some other such method, would skew the results from existing implementations. We can instead extract all the parts of the image that are not part of the ignorable regions. Our goal is thus to create a set of images that, together with the ignorable regions represent the entire image, as shown in Equation (25).

$$\{Image\} = \{Extracts\} \cup \{IgnorableRegions\} \tag{25}$$

However, there exist some cases where the ignorable regions do not allow for the extraction of a region that is large enough for analysis. Consider the case where two

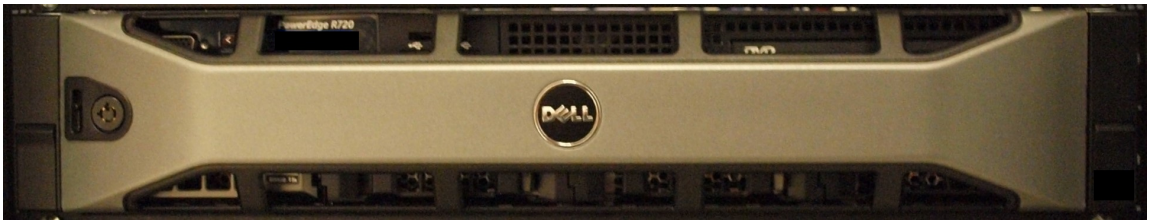

Figure 18: Dell PowerEdge R720 with ignorable regions filled in black (top-middle and bottom-right)

ignorable regions are separated by a column with a width of one pixel. A column of one pixel width is not amenable to shape matching algorithms as there is not enough information to compare. Therefore we must allow for the possibility that some small amount of the image will be discarded when we are creating the extracted images, reflected by Equation (26).

$$\{Image\} = \{Extracts\} \cup \{IgnorableRegions\} \cup \{DiscardedRegions\} \tag{26}$$

When a human examines an image with the ignorable regions blacked out, as shown in Figure 18, typically they can easily identify the areas that should be extracted. It is desirable to extract the set of largest regions possible, while being computationally efficient. There currently are no published algorithms for attempting to find the extract set computationally. I propose an algorithm that iteratively searches an image for these regions.

First, the pixels that belong to the ignorable regions are noted, either through a pixel mask or other facility. The image is then searched, from the top left corner of the image, left to right, row by row, until a pixel belonging to an ignorable region is detected. The pixel at the start of the search will be the top left corner of the region being extracted, labelled by (c_{left}, r_{top}). The first detected pixel that belongs to an ignorable region will be the column of the right outer edge of the extracted area, labelled c_{right}. Once an ignorable pixel has been detected, the algorithm then searches down the columns from c_{left} top to bottom, column by column, until either another

ignorable region is detected, or the bottom of the image has been reached. Once the entire area between c_{left} and c_{right} has been searched, the shallowest ignorable region, or the bottom of the image if there is no ignorable region below that area, is the row specifying the bottom of the image, r_{bottom}. After the four bounding sides of the region have been detected, the area can be extracted. The extracted area is then added to the mask specifying the ignorable regions. The search is then repeated, starting at (c_{right}, r_{top}). The process continues until the set of ignorable regions contains the whole image.

The extracted regions are then tested to determine whether they are of sufficient size for further matching tests. If the extracted region has a row or column dimension that is less than 1% of that of the entire image, it is discarded. If the extracted region has an area of less than 5% of the area of the larger image, it is also discarded. Finally, if the ORB feature point detector, as described below, does not find any keypoints on the image, it is discarded.

Pseudo code for the algorithm to extract valid regions is shown in Algorithm 6.

The above algorithm provides a set of images that can be used for comparison. The next step is to determine the attributes of the images that will be used for comparison. Both shape and colour are attributes that have historically been used for such comparisons [15, 10, 33]. Additional attributes, such as texture, depth maps, and others, could also be used in combination with shape and colour. For our initial purposes, those two attributes will provide a reasonable representation. For each image extracted by the algorithm shown in Algorithm 6, we will apply the following set of algorithms to generate a score indicating how similar two images are.

Shape matching typically consists of comparison based on edges or feature points, also known as contour-based or region-based matching, respectively [48]. However, feature point algorithms in general have been found to have higher performance [4]. For this reason I have chosen to use a feature point detector. Oriented FAST Rotated BRIEF (ORB) is such a feature detector, and has been shown to provide high quality results, with high levels of performance. ORB can be applied to both the template image, and the image it is being compared to generate the keypoints that signify important features within the images. The keypoints between the two

are then matched to one another using the K-Nearest Neighbours algorithm. This is the standard method of performing image recognition based on feature detection.

Although this creates a list of the points with their most likely matches, we must determine which of those points is a credible match. Multiple methods, such as accepting the closest neighbour found, or testing the nearest neighbour using a distance ratio, have been proposed to determine this issue [28]. The items that I aimed to detect with this procedure often had recurring patterns, which can result in many potential matches between keypoints. For our requirements it is most important that we be certain of the match, and that the ambiguity of a certain feature matching a given point is minimized. That is, there exists only one point that closely matches that keypoint in the corresponding image. For this reason, the nearest neighbour distance ratio was used with a relatively high ratio of five. Thus, only points that are less than five times the distance to the next closest match are accepted.

The above method will determine the list of keypoints that match one another from the two images, however, this does not provide a score that may be used for simple comparison when testing other images. The Least Median of Squares (LMedS) method provides a listing of the keypoints that are deemed as inliers when matched using ORB. To determine a score, I propose scoring the shape matching based on the ratio of keypoints that are inliers compared to those that are not.

Pseudo code for the algorithm to score the shape comparison is shown in Algorithm 7.

To determine the colour matching score when comparing the two images I first relied on colour indexing techniques. Once the colours were indexed into histograms, the histograms can then be compared to one another using correlation. This method forms a common base for many implementations of matching colour in computer vision [27, 15, 35].

However, colour indexing experimentally proved to have mediocre results. The histograms in the templates did not contain the noise and surrounding areas and differing conditions faced by images taken in the field. Even when attempting to take into account lighting conditions, those issues resulted in histograms that differed widely. As an alternative, colour clustering was used. Colour clustering uses the k-

means clustering algorithm to group similar pixels together and determine the colour centroids that make up a particular image. These centroids tended to provide a better abstraction between the various types of images being compared. The centroids can then be compared based on the Euclidean distance between the centroids from another image. The distance can then be used as a similarity score. The high level algorithm is shown in Algorithm 4

The scores generated by shape matching and colour matching are then combined as a weighted average. Depending on the application domain, the two component weights can be adjusted accordingly.

The second set of metadata, the features of a template, also provide key information that should be used in the matching process. These features can be isolated from both the template image and the image being analyzed. To allow from some variation, an appropriate buffer around the desired region on the image being analyzed should be used. This allows for different scales between the two images and a small amount of rotation. The two extracted images can then be analyzed using the same methods for the larger image, excluding the need to remove ignorable regions.

For each of the images extracted to exclude ignorable regions, the scores are combined in a weighted average based on how big the extract is in comparison to the entire image. This process determines the overall score for the overall image. If this score meets a minimum threshold, the features can then be compared. The final score is a weighted average where 50% of the score is made up of the score for matching the overall image. The remaining 50% is then determined by the feature match scores. This provides a single number that gives a score for how comparable two images are.

The scoring is shown by Equations (27) to (29),

$$S_{total} = 0.5 \cdot S_{overall} + 0.5 \cdot S_{features} \tag{27}$$

$$S_{overall} = x \cdot S_{shape}(I, T) + (1 - x) \cdot S_{colour}(I, T) \tag{28}$$

$$S_{features} = \frac{1}{n} \cdot \sum_{i=0}^{n} S_{overall}(I, F_i) \tag{29}$$

where S is a score,
x is a percentage value between 0 and 1,
I is the image extract being analyzed,
T is the template being analyzed,
F_i is the ith feature of the template,
n is the total of features for the template.

5 Experimental Evaluation

5.1 Overview

To test the proposed approach, I worked with a Data Centre Infrastructure Management (DCIM) company. Data centres are large buildings that are full of equipment shelving, known as server racks. The server racks hold computers, networking equipment, power management devices, and more. Data centre equipment is often being expanded and replaced. As these upgrades are performed, it is necessary to efficiently plan where to add new equipment to make the best use of space and resources. To plan effectively, DCIM companies require an accurate understanding of the existing data centre equipment layout, even when new components are added and exchanged week-to-week by various employees and contractors.

Currently, the DCIM company uses a manual process to inventory their warehouses. An employee will walk through the warehouse and record the equipment, or the changes to equipment, in each of the server racks. This is a very labour expensive process and is prone to human and transcription errors. By using an automated computer vision system, the DCIM company will be able to dramatically improve their operations and services.

This section provides a detailed explanation of the effectiveness of the proposed approach in identifying the individual pieces of equipment.

5.2 Prototyping Environment

5.2.1 Running Environment

The running environment for the development of this experiment was a Macbook Pro (Early 2015). It is expected that eventually this research work will be used in a production environment with a dedicated server. It was desirable that the system can function with reasonable performance without dedicated hardware such as graphics cards optimized for computer vision tasks. The specifications of the running environment are shown in Table 1

Device	Macbook Pro (Early 2015)
OS version	Microsoft Windows 10
CPU	2.7 GHz Intel Core i5
Number of CPUs	4
Memory	8 GB 1867 MHz DDR3
Storage	50 GB
.NET Version	4.5
C++ Version	C++14
IDE	Microsoft Visual Studio 2012
Database	MongoDB 3.1
OpenCV	OpenCV for Windows v3.1

Table 1: Experimental Environment Specifications

5.2.2 OpenCV

OpenCV is the dominant open source computer vision library. The library is widely used in industry and academic settings [5]. The library offers a wide range of vision facilities, with a focus on practical applications. These facilities include pattern recognition, camera calibration, robotic vision, stereo vision, machine learning, and more [5]. OpenCV has been publicly available and in continuous development since 1999. The software played a role in the earliest autonomous vehicle systems, including "Stanley", the winner of the original DARPA Grand Challenge desert robot race [5, 45].

The library is written primarily in C and C++, although there are wrappers implemented in other languages, such as Python, Java, C#, and more. Of the many available languages, the C/C++ version was used due to the completeness of the public API (some wrappers are not feature complete), and the efficiency. Wrappers of the original library implicitly add extra computation.

Multiple factors make OpenCV an ideal candidate for usage in computer vision projects. The library is robust, there are a wide range of documentation and tutorials available online, and the expert community online is very active. The usage of this library resulted in many hours of work being saved by removing the need to implement existing algorithms unnecessarily. It is for these reasons that OpenCV

was used in this research.

5.2.3 Programming Languages

Since the OpenCV C/C++ implementation was chosen, the primary languages used in developing the experimental solution was C# and C++. C# was used for the majority of development to take advantage of higher-level language constructs. C# is also well supported for interaction with 3rd party systems, such as MongoDB.

Separate C++ programs were developed to perform interactions directly with the OpenCV library and the images. The main C# application launches the C++ programs via the "Platform Invoke (P/Invoke)" method. Communication between the two languages is handled via command line arguments and standard in and standard out.

5.3 Testbed Description

The testbed system is first preloaded with an image library of templates for analysis. The system can then accept an image of a server rack as input. Next, the rack image is prepared for analysis and compared against the image library. The scores from the comparisons are then ranked to provide an ordered list of potential matches. These steps are described in detail here.

5.3.1 Preprocessing

In order to prepare the system for analysis some preprocessing is required. The system requires a series of template images that define the potential pieces of equipment in an image. These template images required a data model, in the format of a JSON file listing their description, as specified in Chapter 3. An example of one of these files is shown in Figure 15. Where possible, the template images were acquired from the data centre with the equipment completely unplugged and powered down. When this was not feasible, such as when equipment could not be powered down for

operational reasons, images from manufacturer websites were used. The accompanying JSON file was filled out by the DCIM company with the help of some simple, web-based tools for determining the coordinates and size of ignorable regions and feature locations. When loaded into the system, the image was copied to a local file folder, and the JSON object was inserted into a MongoDB database collection.

5.3.2 Calibration

Images of server racks were obtained from a data centre. Due to the limited space in the data centre, a camera with a large field of view was required to capture the entire server rack. However, in order to provide the large field of view, GoPro uses what is called a "fish-eye" lens. This lens is circular, and shows a distorted and curved image, as can be seen in Figure 8. The distortion by such lens vary depending on where an object is within the image. An object at the centre of the image will have close to zero distortion. An object at the end of an image will have a lot of distortion, with a stretched appearance. This is problematic when comparing the images to template images, as the template images will not be distorted. Further, distortion cannot be applied to the images to mimic the GoPro lens because the distortion varies throughout the image. Therefore, the images from GoPro cameras must be calibrated and undistorted.

In order to calibrate for a given camera multiple test images must be taken. These images involve attaching a print out of a checkerboard pattern to a flat surface. An example of a calibration image is shown in Figure 19. Multiple pictures are then taken with the pattern positioned at various angles, locations, and distances. These pictures are then used by a calibration algorithm to determine the distortion by the camera.

Using the method proposed by Zhang [49], the parameters provided by Equations (1) to (3) are used to undistort the image, resulting in an image that has close to zero distortion.

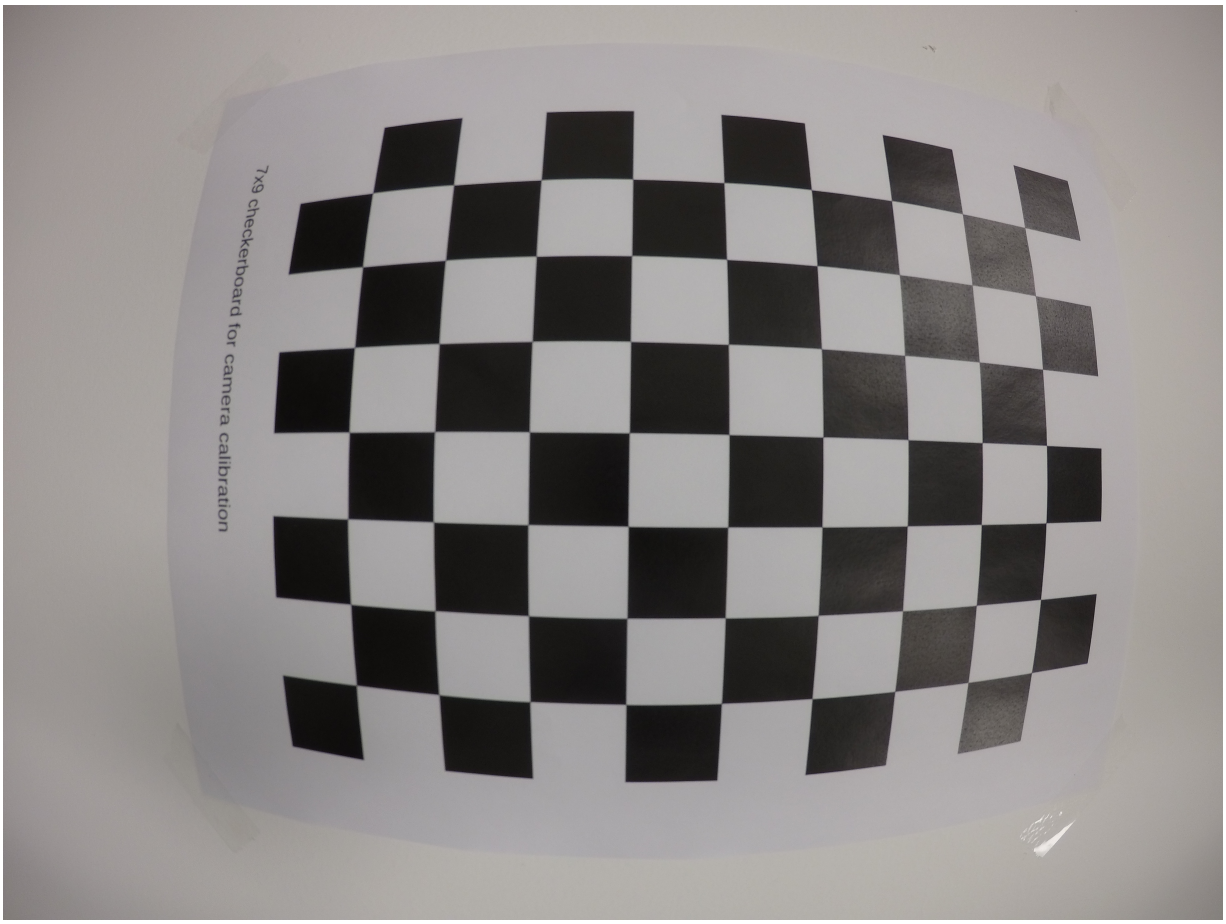

Figure 19: Example of a calibration image

5.3.3 Edge Detection

The first phase of the proposed approach is to identify the individual items on the shelf. In the demonstration system, that requires identification of the individual pieces of equipment in the server rack. This was conducted in two phases. First, the primary server rack in the image is identified. Second, the individual pieces of equipment within that server rack are identified. The below process is completed once for each server rack image that is input into the system.

Server Rack Detection As can be seen in Figure 20, a server rack can be identified as the rectangle outlined by the two vertical dark grey bars that support the equipment within the rack, and the two horizontal bars dark grey bars at the top and bottom of the rack. Due to the camera field of view, an image of the server racks may contain more than just a single server rack. The server racks to the left and right of the main rack may be partially visible as well. Additionally, the camera may capture areas above and below the main rack. Anything in the image outside of the server rack is not useful information in determining the identity of the equipment within the rack. In fact, this area could interfere with matching process. To remove the excess areas from the image, we must first detect the location of the four edges of the server rack. Once we have detected those edges, we can crop the image to remove anything outside the rack.

To detect the dark grey bars in the image the same algorithm is used on each bar. It is assumed that all images of the server racks will be taken with the same type of camera. Every server rack within a data centre is typically the same standardized size. Based on these facts, I determined ahead of time that the width (for vertical bars), and height (for horizontal bars) of the rack sides to be 150 pixels.

Due to the solid, dark grey colour of the bars in the rack, we can detect them by using thresholding and searching for a continuous black region. First, we threshold the image using Otsu's Method [31]. Then, we search for the left vertical bar of the rack. This is done by searching from left to right until a column of pixels that are all black is found. This is repeated until a set of 150 entirely black columns are found.

Figure 20: Rack 2, with the perimeter of the rack outlined in green

This region of 150 black rows are deemed to be the left side of the rack. It is also possible that there could be occlusions, such as hanging cables, that appear within the region containing the server rack bar. There may also be glare from overhead lighting that would result in some pixels within the region being thresholded to white, instead of black. To account for these cases, we specify a threshold that the region must meet. Through experimental trial and error, I determined that up to 20% of the region's pixels may be white. Therefore, the region must satisfy Equation (30), where the left hand side is the number of black pixels in the region spanning columns x to $x + 150$, and the right hand side is the threshold percentage t multiplied by the total number of pixels in the region. Finally, we assume that the server rack we are analyzing will be the main object within the image. Based on the existing images, we know that the left side of the server rack will be within the first vertical quarter of the image. We therefore limit our search to columns within the first quarter of the image. Algorithm 8 shows the pseudo code for the completed algorithm.

This same algorithm is applied to find the right side of the image. Instead of starting our search from the left-most column, the search begins from the right-most column. For the horizontal bars we search for 150 entirely black rows, instead of columns. For the top horizontal bar, the search starts from the top of the image. For the bottom horizontal bar, the search starts from the bottom of the image. In the case of the top horizontal bar, the region may be almost entirely thresholded to white, due to overhead glare. To account for this, we repeat the algorithm but instead search for entirely white rows instead of entirely black rows. We use the same threshold of 20% to specify the allowable number of black pixels. This is shown by Equation (31).

$$r_{x \to x+150}(p_{black}) > t \cdot r_{x \to x+150}(p_{all}) \tag{30}$$

$$r_{x \to x+150}(p_{white}) > t \cdot r_{x \to x+150}(p_{all}) \tag{31}$$

Equipment Detection Once the server rack has been isolated, the next stage of the process is to isolate the individual pieces of equipment within the server rack. This process is aided by the fact that we know that pieces of equipment generally have solid, straight edges, and that they are at least 1RU in height. Based on analysis of the server rack images provided, it can be determined that 1RU is roughly equivalent to 167 pixels in height.

First, a standard Canny edge detector is applied to the isolated server rack to find all edges within the image, as shown in Figure 21 [7]. Canny edge detection will find a large number of edges in the image, with many different angles and shapes. To determine straight lines within the image a Hough transform is applied. In this case, an enhancement of the Hough transform, the Progressive Probabilistic Hough Transform (PPHT) is used. This enhanced version was chosen because it has been shown that its efficiency improves significantly on the original, with only a slight decrease in quality [25]. The PPHT algorithm will detect many lines, often short in length and at various angles. However, since we know that the edges of the server rack will be horizontal within the image, we can isolate those lines with an angle of zero degrees. To account for possible tilt in the image, we allow the isolated lines to have an angle between -5 and +5 degrees. This step is shown by Equation (32), where L is the set of lines such that the angle of the line is between $5°$ and $-5°$. This will result in a large number of short line segments that are essentially horizontal.

Now that we have the horizontal lines within the image, we must determine which rows within the image are most likely to contain a long horizontal line that aligns with the edge of a piece of equipment. To achieve this, we first measure the maximum number of detected lines that start or end on a given row. This is shown by Equations (33) and (34), where $count(r_y)$ is the count of lines in set L such that the row the line is on, l_y is equal to the specified row r_y, and t is the threshold determined as the maximum number of lines found by $count(r_y)$ for all rows. Next, we discard any lines that are on rows with less than 30% of the maximum number of lines on a row.

This prevents any spurious horizontal lines that have been detected in the background of a server rack, for example, where there is an empty space in a server rack. Finally, we search, from top to bottom, over the image 1RU (167 pixels) at a time.

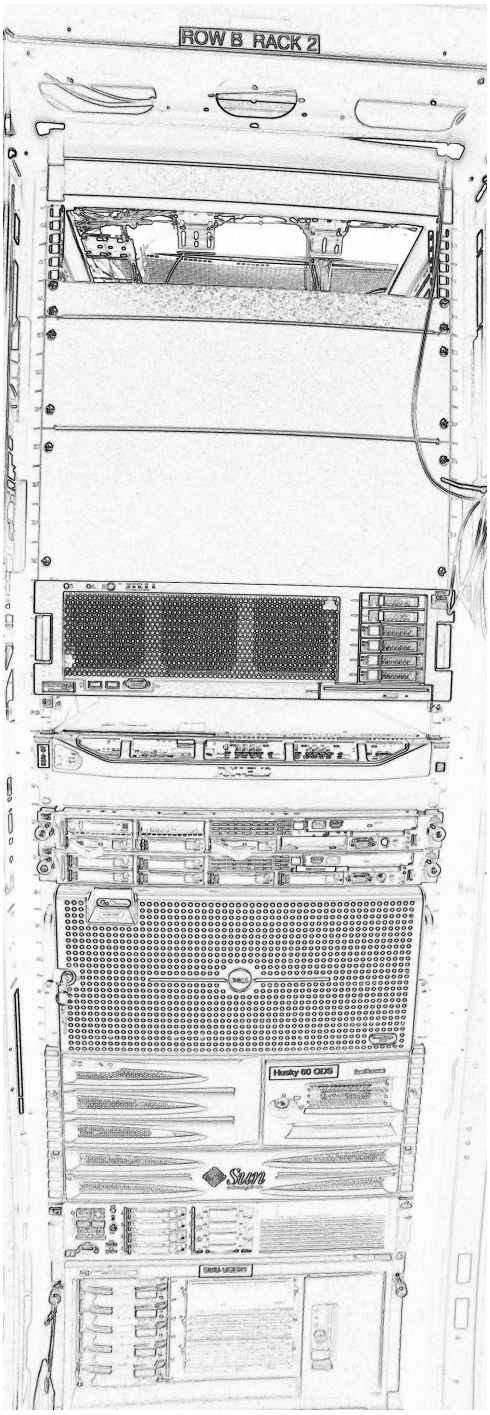

Figure 21: The results of applying Canny edge detection to a server rack

Within that window, we only keep the row with the most line segments. The final set of rows are shown by Equation (35), shows the final set of rows, R. The output of the algorithm is shown in Figure 22.

Algorithm 9 shows the pseudo code for the completed algorithm.

$$L = \{l \mid (angle(l) < 5^\circ \quad and \quad angle(l) > -5^\circ) \quad \} \tag{32}$$

$$count(r_y) = \sum l \in L \quad s.t. \quad l_y = r_y \tag{33}$$

$$t = 0.3 \cdot max(count(r_y)) \tag{34}$$

$$R = \{r_y \mid count(r_y) > t \quad and \quad count(r_y) > count(r_y + 167)\} \tag{35}$$

5.3.4 Region Matching

The edge detection portion of this solution isolates the images for analysis. The region matching portion compares the template images to the individual equipment images output by the edge detection process. Region matching is applied once for every template image against every extracted equipment image. Region matching accepts as input a template image from the image library with its accompanying data model, as described in Chapter 4, and an extracted image from the edge detection process. The output of the process is a set of scores that approximate how closely the image extracted by edge detection resembles each of the template images.

Extracting Valid Regions Given two images, one template image and one extracted image, the first obstacle to performing image matching is how to use the common methods for comparing images while ignoring parts of the region that have been marked as ignorable. Simply removing the ignorable regions from the image by zeroing out the pixels in that region would skew existing image comparison algorithm implementations. To avoid re-implementing existing algorithms to ignore the speci-

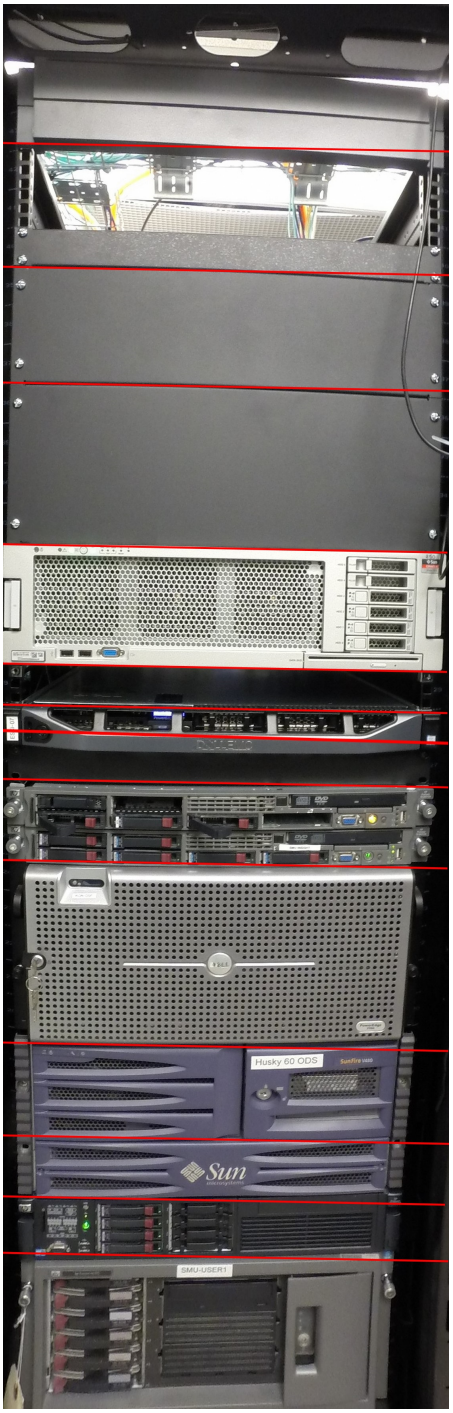

Figure 22: The lines separating pieces of equipment in a rack, as found by the edge detection algorithm

fied regions, we can instead break our images into separate parts that do not include the ignorable regions. The sub-images from the template and the extracted images are then compared to one another using existing algorithms and implementations. However, cropping images manually to remove the ignorable regions is not feasible in a production system. Currently, there are no existing algorithms that describe how to break an image into composite sub-images, while deliberately ignoring specific regions within the image. The following algorithm is proposed to efficiently find the sub-images.

Since colour is not key to determining the sub-images, we convert any colour images to grayscale, for simpler processing. Starting with the matrix of pixel values representing an image, we must first mark the ignorable regions in the matrix. We will use the value of 0 (black) for every pixel that is located within the ignorable regions. In order to not confuse a pixel that is actually black with a pixel that is in the ignorable region, we first add one to all of the values in the matrix.

The algorithm follows the simple goal of searching from the starting point, the top-left corner of a new sub-image, as far as possible to the right, and as far as possible to the bottom to determine the area to extract. The algorithm starts at the origin of the image, in the top-leftmost corner. The search to the right continues until it meets the edge of the image, or the edge of an ignorable region. The search to the bottom will similarly continue until it meets the bottom edge of the image, or the edge of an ignorable region.

The points where the search ends determine the edges of the sub-image. Once a sub-image has been found, the coordinates are recorded, and all pixels in the region are marked as 0. This way the sub-image region is not included in any future sub-images. This search continues until the entire image has a pixel values of 0.

Depending on the layout of the ignorable regions, some sub-images will inevitably be too small. A sub-image is considered too small for further analysis if it has dimensions that are less than 1% of either corresponding dimension for the entire image. Figures 18 and 23 show an example image with ignorable regions drawn, and the corresponding sub-images found by the algorithm.

While other potential algorithms were considered, this algorithm has the benefits

Figure 23: The five valid regions extracted from the Dell PowerEdge R720 template

of simple implementation and simple computational time.

Algorithm 10 shows the pseudo code for the completed algorithm for extracting valid regions from an image with ignorable regions.

Shape Matching The first of two components we use for comparing two images is shape. After applying the algorithm to extract the valid regions from the template image, and extracting the corresponding regions from the extracted image from the server rack, the next step is to detect the features of the two images. I then detect the homography between the two sets of features which will also provide a set of inlier points. A score is computed based on the ratio of outlier points to all points.

To detect features, the Oriented FAST, Rotated BRIEF (ORB) algorithm is applied to the two images being compared. This algorithm was chosen due to its speed and quality of results. The ORB algorithm determines a set of features, or keypoints, in each of the two images. The algorithm then describes each of the features. These descriptions are called descriptors.

The two sets of descriptors are compared using the k-Nearest Neighbours (kNN) algorithm. The kNN algorithm provides a ranked list of possible matches for each keypoint. Only matches that are significantly stronger than the second closest option are kept. Stronger is defined as having a Hamming distance less than five times the distance of next closest match. This variability in distance suggests unambiguity, and thus confidence, in the first match.

The least median of squares (LMedS) method is used to find the homography between the two remaining sets of keypoints. LMedS will only work when more than 50% of the points are inliers [26]. Therefore, if LMedS fails we discard the match as

"not a match". Otherwise, LMedS will output a list of points that are inliers.

The size of the list of inliers outputted by the LMedS algorithm can then be used to determine a score for the similarity of the two images. If there is a high proportion of inliers compared to the initial list of keypoints, then the two images are determined to be a good match. If there is a low proportion of inliers, the two images are not as good of a match. The score is computed as the ratio of outliers to the total list of keypoints, or one minus the ratio of inliers to the total list of keypoints. The closer a score is to zero, the better the match is deemed.

Algorithm 11 shows the pseudo code for the completed algorithm. Figure 24 shows examples of matching part of both a correct template, and an incorrect template to an image extracted from a server rack.

Colour Indexing A second component that can be compared between a template image and the extracted image is the colour. One method of doing so is via colour indexing. Colour indexing provides a simple and efficient method for determining the similarity of pixel values between two images.

This process takes two images as input, the template image, and the extracted image from the server rack. The two images are then converted from the standard RGB colour space representation to the Hue, Saturation, Value (HSV) colour space. The hue and value channels are then isolated from the two images. The saturation channel is ignored as it is the most susceptible to differences in lighting conditions such as glare.

The histograms of the two remaining channels in the two images are then computed. The histograms are normalized to values between zero and one in order to compare the two histograms. The difference between the two histograms is computed using the Correlation method. Using this method, a perfect match will receive a value of one. An item that does not match completely will receive a value of zero. The pseudo code algorithm is shown in Algorithm 12.

Although colour indexing provided decent results when using template images that had been taken in an operational environment (i.e. the same environment that the images of the server racks were taken), they did not do as well when using images

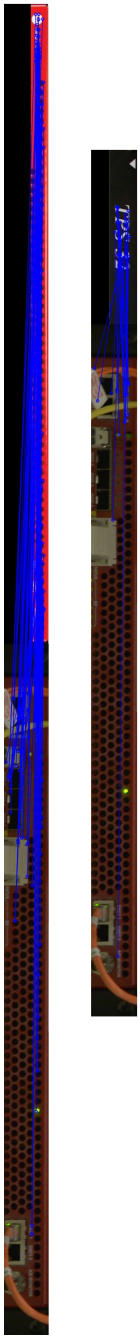

Figure 24: Lines drawing the keypoint matches between a correctly matching template (left), and an incorrectly matching template (right)

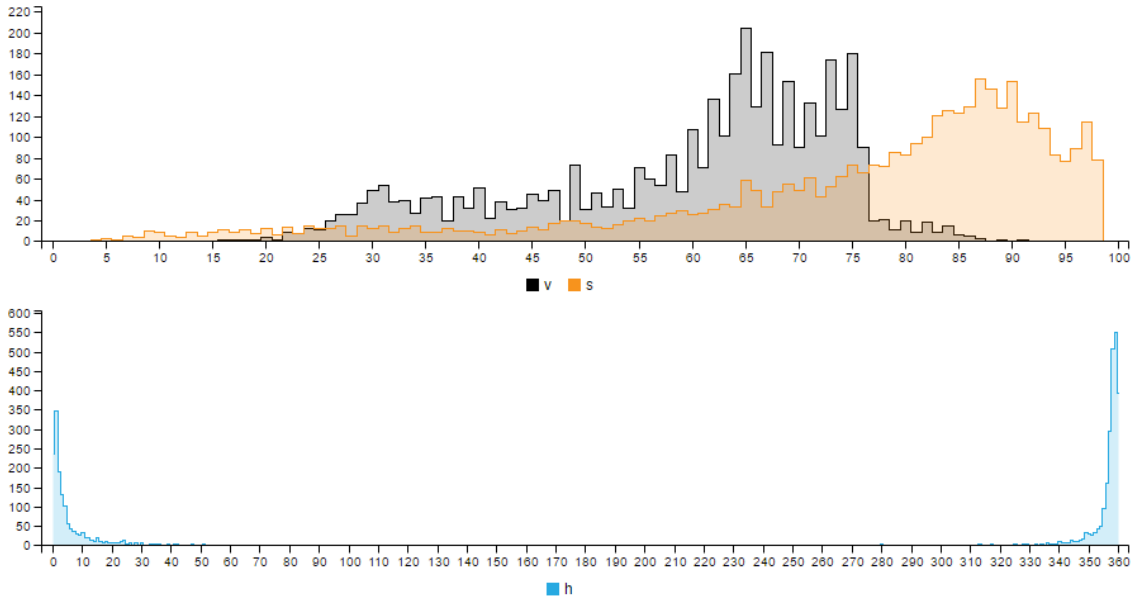

Figure 25: The HSV histograms generated for the Apex 1000 template image [21]

obtained from other sources. In these instances, the histograms, as can be seen in Figures 25 and 26 differed significantly. The areas surrounding the extracted image have a significant impact on the topography of the histograms, even when removing the saturation channel. To improve upon colour indexing, colour clustering was used.

Colour Clustering Colour clustering provides representative colours that can be used for comparison between images. Colour clustering works by applying the k-means clustering algorithm to the pixels of the image. By clustering both the extracted image and the template image, we can find the colours that best represent the depicted equipment.

To determine the number of clusters used to cluster each image, a small sample of tests were run on a number of representative images. For each image, clustering was performed with a varying number of k specified clusters. Based on these tests, five was chosen as the optimum number of clusters.

Once the five representative pixels have been found for each image they can be compared to one another running through an iterative process to determine the

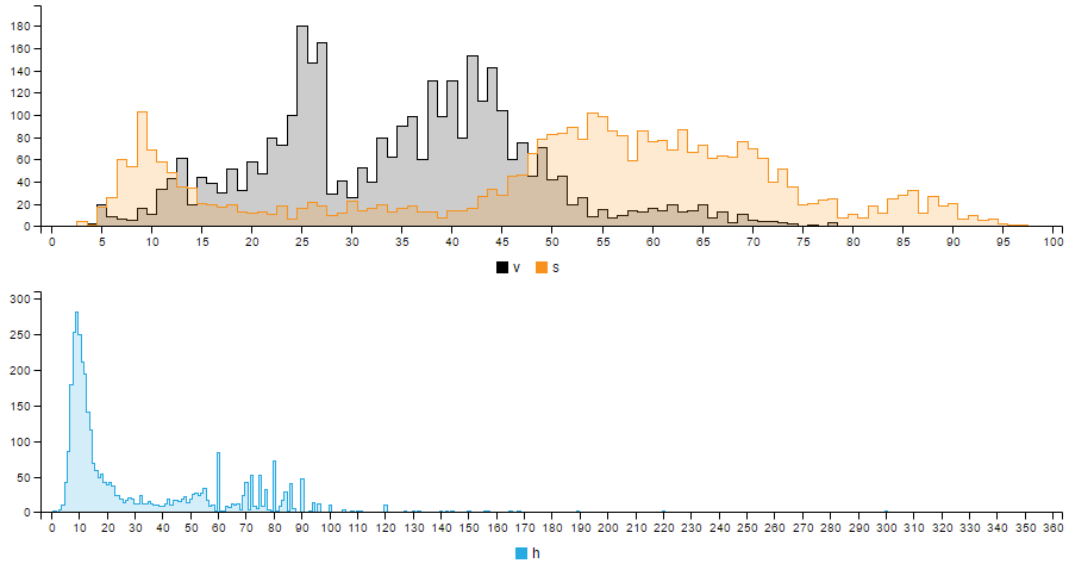

Figure 26: The HSV histograms generated for an extract image of an Apex 1000 server [21]

closest pixel between the two sets. The Euclidean distance between each matching set of points is then summed to determine the overall difference between the colours in the two images. The iterative process of determining the best match loops through the points, finding the best match, and removing it from the list of possible options for other points is shown by Algorithm 3. This process is repeated for all permutations of the source set that finds a match. All permutations are generated via Heap's algorithm [12]. All permutations are tested to ensure that the the optimal set of pairs are found. Based on experimental testing, the images typically had a distance of between 100 and 1000. Therefore, each total distance is divided by 1000 to determine an overall colour score, where the lower the score is, the better the match. The entire pseudo code algorithm can be seen in Algorithm 13.

Feature Matching Once the entire image has been compared using shape matching and colour indexing, each of the features in the template image are compared to their expected position in the server extract image. First, the location of the feature

Algorithm 3 Pseudo code algorithm for comparing two sets of points to find the closest matches

```
1: while exists new permutation of imageCentroids do
2:    workingSetCentroids = templateCentroids
3:    for imageCentroid in imageCentroids do
4:       for templateCentroid in workingSetCentroids do
5:          currentDistance = euclideanDistance(imageCentroid, templateCentroid)
6:          if currentDistance <closestDistance then
7:             closestDistance = currentDistance
8:             closestIndex = templateCentroid index
9:          end if
10:      end for
11:      closestDistanceSum = closestDistanceSum + closestDistance
12:      remove closestIndex from workingSetCentroids
13:   end for
14:   if  closestDistanceSum <currentClosestDistanceSum then
15:      closestDistanceSum = currentClosestDistanceSum
16:   end if
17: end while
```

within the template image is cropped and extracted. The image extracted from the server rack may vary in size, especially if the template image was taken by a different camera. To account for this difference, the extracted image is first scaled to similar dimensions. The rectangles bounding features on template images are very precise. To account for slight differences in the position of the piece of equipment within the image, we allow for a 15% buffer along all sides of the area where the feature should be in the server extract image. This enlarged area is cropped and extracted for comparison to the feature extract from the template image. The processes described in Shape Matching and Colour Indexing above are repeated for these two images. This process is repeated for every feature of the template image.

Matching Multiple A single area extracted by the equipment detection stage of the algorithm may in fact contain more than one piece of equipment. This can occur when the lines at the edges of the piece of equipment are occluded, or when another

line near the piece of equipment is simply more defined. In order to account for this, I run a single extracted image through the entire process multiple times as needed.

First, the size of the extract is considered. If the extract is at least as large as approximately 2RU, it is eligible for having multiple pieces of equipment. Once a piece of equipment has been detected on the image, we can determine it's location by drawing a bounding rectangle around the inlier keypoints. The keypoint in the top-right corner of the bounding rectangle can be used as an approximate location for the piece of equipment in the image. Based on the location, and the size of the piece of equipment detected, it can be estimated if there is enough room in the image such that another piece of equipment could be found.

If the location of the previously found piece of equipment is near the top of the image, that area is removed, and the bottom portion of the image is run through the algorithm again. If the location of the previously found piece of equipment is near the bottom of the image, that area is removed, and the top portion of the image is run through the algorithm again. Finally, if the location of the previously found piece of equipment is in the middle of the image, that area is removed, the image is split, and both the top and the bottom portions of the image are run through the algorithm again.

For each additional piece of equipment from a sub-region (as outlined in the previous paragraph) that is detected, an additional array is added to the data model for that "Server", as can be seen in the JSON in Figure 10. The arrays are included in order, such that the first array includes results from matching the top of the image, and the last array includes results from matching the bottom of the image.

Scoring A cumulative score is compiled using the shape matching and colour indexing results. This score is used to provide a score of how similar the extracted image is to a given template image.

The shape matching algorithm outputs a value describing the similarity of shapes between the two images. The closer the value is to zero, the closer the two images are in similarity. However, the colour indexing algorithm outputs a value where the closer a value is to one, the more similar the two images are. To combine these

scores, the colour indexing score is subtracted from one, so that the closer the value is to zero, the closer in similarity the colour between the two images will be. It was decided that matching the shape of the piece of equipment is more important than matching the colour of the piece of equipment. As a result, the scores are combined with a weighting of 55% for the shape of the equipment, and 45% for the colour comparison.

For all of the features, this score is then multiplied by the weight of the feature as described in the template image's data model. All of the feature scores are combined as a weighted average into a single feature score.

The score for the entire image and the feature score are then combined, with each component weighted at 50%. This score is used for ranking the extracted image's similarity to the template. The equations used are shown by Equations (36) through (38).

$$S_{total} = 0.5 \cdot S_{overall} + 0.5 \cdot S_{features} \tag{36}$$

$$S_{overall} = 0.55 \cdot S_{shape}(I, T) + 0.45 \cdot S_{colour}(I, T) \tag{37}$$

$$S_{features} = \frac{1}{n} \cdot \sum_{i=0}^{n} S_{overall}(I, F_i) \tag{38}$$

Optimizations There exist within the above algorithm some optimizations that can be used to improve the analysis runtime. There are three types of optimizations that are readily apparent, short circuits, caching, and parallelization.

Short circuits can be used at various steps where it is obvious, based on the results of the previous step, that the template will not be a match for the image. Generally, this involves determining whether a score is below a predefined threshold. Examples of this optimization can occur after shape matching, before feature matching (after colour matching), and during feature matching if the image has already failed to match a given percentage of features.

Caching can be used in cases where the data is independent of the two images

being compared. For example, all feature extracts from the template images can be saved and re-used, as well as saving all images in memory for faster access. Another example is the results of time intensive algorithms, such as caching the keypoints detected by the ORB algorithm, or the clusters found via K-means.

Finally, the algorithm is highly parallelizable. Since multiple images must be compared to multiple other images, a thread can be created for each comparison (as the system allows). On the test equipment, it was found that one thread per CPU core at a time was optimal to limit thrashing. Further, we can ensure that the order of templates compared in each thread differs, to prevent multiple threads from computing values that will be cached at the same time.

5.3.5 Final Ranking

After an extracted image has been scored against each template image, the matches can be ranked. The matches are ordered by score from lowest to highest, where the lowest score means that the match was most similar.

5.3.6 Evaluation Tasks

This demonstration system was analyzed in two stages. The first stage of analysis aimed to only measure the quality of the region matching algorithm. The second stage of analysis measures the quality of the entire system.

For this demonstration system, the desired outcome was to provide a limited set of options to a human operator. The operator could then verify and select the correct match. Therefore, when evaluating the system it was important that the correct match appear within a short list of limited options. The options listed for each piece of equipment were examined with regards to how often the correct match appeared as the first option, within the top five options, and within the top ten options listed. Identifying areas where no piece of equipment was present was outside the scope of the prototype system, and so those regions were ignored in calculating the matches.

To test the region matching system, pieces of equipment were manually cropped from the existing server racks. These images were manually identified and then run

Image Type	Number of Images
Rack Images	8
Pieces of Equipment	95
Equipment Templates	47

Table 2: Test Data Summary - Phase 1

through the region matching system. The results were then analyzed to determine the ranking of the correct match.

For evaluation of the entire system, the rack images were run through all stages of the protoype system, including preprocessing, extraction and region matching. The results were then analyzed to determine the ranking of the correct template to the piece of equipment.

5.4 Testing Data Description

The test dataset was composed of rack images and templates as described in Chapter 3. The test dataset was gathered in two phases, the first used a camera with a narrow field of view. The rack images captured by this camera required manual post-processing to stitch multiple images into a single composite rack image. Template images for the first phase were obtained from a variety of sources, including the Internet, product manuals, and samples from the rack images. The second phase of test data was captured using a camera with a wide field of view. The wide field of view camera was able to capture a single server rack in one image. Template images for the second phase were entirely sampled from the rack images.

The test data is summarized by Tables 2 and 3.

Two different tests were performed on each dataset. The first test was purely a test of the matching algorithm. The pieces of equipment were manually cropped from the rack images. These images were then compared directly to the template library. The second set of tests tested the entire system from end-to-end.

Image Type	Number of Images
Rack Images	112
Pieces of Equipment	406
Equipment Templates	74

Table 3: Test Data Summary - Phase 2

5.5 Manual Extract Test Results

Tables 4 through 7 summarize the manual extract test results. Table 4 shows a summary of the extract test results for the images from the first phase. A notable feature of this data is that in cases where a piece of equipment was not ranked at all, the equipment did not rank for all instances of that piece of equipment. When a piece of equipment was ranked, all instances of that piece of equipment were ranked. This is likely due to an issue with the template image of that piece of equipment. If an image is not ranked at all, it is typically due to a poor score during feature matching. In these cases, the algorithm is unable to match a sufficient number of points, or, the points that it does match perform below the threshold. It is also possible that the ignorable regions are too broad, leading to a limited number of features.

Of the pieces of equipment that ranked, we see only seven pieces of equipment that consistently ranked within the top 10. This is further demonstrated by Table 13. Here we see that nearly half of all images ranked within the top ten. However, the correct image ranked first in only one instance.

The results from the second phase in Table 6, which used a larger set of equipment and images, show slightly poorer results. Despite the near doubling in number of types of equipment, the absolute number that ranked within the top ten is only slightly higher, with ten. In the phase two results, as shown in Table 7, we see that with a larger set of images, a greater number of images ranked as the top choice. However, there are a similar number of images within the top ten for both test datasets. Given the much larger size of the phase two dataset, this results in only a fifth of the images tested being ranked in the top ten.

Equipment Name	Images Tested	Number Correct Template Ranked	Average Rank
ATX MN5	4	4	7.25
Aurora CH3000N	8	8	4.75
Cisco RFGW2	14	14	17.29
Drake VM2410A Modulator	1	1	23
Fujitsu FW7500	2	0	-
Jerrold S450M	1	1	19
Juniper EX4550	5	5	18.4
Juniper MX960	1	0	-
Leitch 6800	1	0	-
Motorola Apex 1000	14	14	9.86
Motorola NE2500	9	9	2.78
Motorola OM2000	8	8	24.88
MPEG2 DSR-4400	5	5	6.60
Nortel NT0H32AH	2	2	19
Nortel NT0H32BF	4	4	13.25
Nortel Optera Metro 5200	4	0	-
Telect GMT 10-10 Fuse Panel	3	3	9
Telect GMT 10-10 Fuse Panel 2	1	1	1
Telect HPGMT15 Fuse Panel	2	2	9
Telect KLM-GMT 4-4 Fuse Panel	1	1	26

Table 4: Results - Phase 1 - Extract Test Equipment Summary

	Number of Images	Percent of Total
Total Images	95	-
Total Ranked	82	86.32%
Top 10	39	47.56%
Top 5	26	31.71%
Top 1	1	1.22%

Table 5: Results - Phase 1 - High Ranking Summary

Equipment Name	Images Tested	Number Correct Template Ranked	Average Rank
A10 Networks AX 3400	2	0	-
ADC FVM-19x700	10	10	27.20
Agilent Spectrum Analyzer E4411B	5	5	18.00
ATX MN1-16	25	25	16.08
CableServ CHAS	13	13	34.08
CableServ CHAS 2	4	4	13.00
Casa Systems C100G	5	5	2.80
Cisco 4506	15	15	1.47
Cisco D9500	2	2	12.00
Cisco UBR-RFSW-3X10 RF Switch	4	4	1.25
Cisco uBR10000	1	1	22.00
Colomachine CM61	4	4	8.25
Electroline TPS MS-100	5	5	16.6
Electroline TPS SL-100	12	12	22.83
Electroline TPS-32	3	3	12.67
Generic 6-port AC Power Outlet	2	2	7.00
JDSU Stealth Sweep Transceiver	1	1	15.00
Juniper MX960	5	5	10
Juniper SRX5600	1	1	1.00
Motorola Apex-1000	29	29	1.62
Motorola ARPD 1000	31	31	25.48
Motorola NE2500	9	9	12
Motorola Receiver	4	0	-
Motorola SE-1010	28	28	23.21
Motorola SEM V8	1	1	1.00
Motorola SEM V8 Cover	8	8	21.00
Nortel OPTera Metro 5200	1	1	27.00
Palmorex PMX 1500	14	14	32.64
PCI Technologies MN2	13	13	6.23
PCI Technologies MN5T	72	72	23.82
PCI Technologies RMS PCI-81L 8-way Splitter	1	1	1.00
PCI Technologies SCN-PP5-20F 5-20 Demark	7	7	12.71
PCI Technologies TSG 4000R	1	1	12.71
RGB Networks SEP48	4	4	24.50
Scientific Atlanta D9510	2	0	-
Scientific Atlanta Prisma DTx	4	0	-
Standard TVM 550 II5	1	1	12.00
Telect 10-10 GMT Fuse Panel	20	20	19
Telect HPGMT15 Fuse Panel	14	14	20.07
Telect KLM-GMT Fuse Panel	21	21	19.57
Trilithic Super Series CT-2	1	1	12

Table 6: Results - Phase 2 - Extract Test Equipment Summary

	Number of Images	Percent of Total
Total Images	406	-
Total Ranked	395	97.29%
Top 10	80	20.25%
Top 5	65	16.46%
Top 1	48	12.15%

Table 7: Results - Phase 2 - High Ranking Summary

Mean	12.07
Median	11.00
Standard Deviation	8.1085

Table 8: Results - Phase 1 - Manual Extract Basic Statistics

5.5.1 Basic Statistical Performance

Tables 8 and 9 show the mean, median, and standard deviation of the test results. In both phases, the mean and median values are outside of the top ten, with slightly worse performance in the phase two dataset. The standard deviation is also quite large, given the size of the template libraries, in both datasets, at 8.1085 and 11.82. These values show that overall, the region matching system alone was not able to effectively limit the number of choices to the top ten. With the larger and more variable dataset from phase two, the mean correct match falls to the bottom of the top twenty.

Mean	18.31
Median	17.00
Standard Deviation	11.82

Table 9: Results - Phase 2 - Manual Extract Basic Statistics

5.5.2 Mean Reciprocal Rank Performance

Mean reciprocal rank is a statistical method of evaluating information retrieval queries [8]. It is best applied to known-item searches [29]. A known-item search is where there is only one correct answer to a query, and the result set is a ranked list. The problem studied can be accurately described as a known-item search.

Mean reciprocal rank is described by Equation (39).

$$MRR = \frac{\sum_{i=1}^{|Q|} \frac{1}{rank_i}}{|Q|} \tag{39}$$

where Q is the set of queries performed for a piece of equipment,
$rank_i$ is the rank of the correct template for query i.

A perfect mean reciprocal rank is 1. If all queries had ranked the correct match at two, the mean reciprocal rank would be 0.5. The lower the mean reciprocal rank, the poorer the results for that set of queries.

Tables 10 and 11 show the mean reciprocal ranks for each type of equipment. Some equipment, such as the Motorola NE2500 and Motorola Apex 1000, perform significantly better than the others. This is may be due to the distinctive colour of the pieces of equipment. There are a number of pieces of equipment that have a perfect score. However, in all cases these pieces of equipment only had one test image. It is likely a larger dataset for these pieces of equipment would result in lower scores.

The mean reciprocal ranks similarly show that the proposed region matching system was unable provide high quality results overall.

Equipment Name	**Mean Reciprocal Rank**
ATX MN5	0.186905
Aurora CH3000N	0.288542
Cisco RFGW2	0.060275
Drake VM2410A Modulator	0.43478
Fujitsu FW7500	-
Jerrold S450M	0.052632
Juniper EX4550	0.05629
Juniper MX960	-
Leitch 6800	0.066667
Motorola Apex 1000	0.140491
Motorola NE2500	0.37037
Motorola OM2000	0.040554
MPEG2 DSR-4400	0.277193
Nortel NT0H32AH	0.058462
Nortel NT0H32BF	0.083802
Nortel Optera Metro 5200	-
Telect GMT 10-10 Fuse Panel	1.000000
Telect GMT 10-10 Fuse Panel 2	0.143056
Telect HPGMT15 Fuse Panel	0.138462
Telect KLM-GMT 4-4 Fuse Panel	0.038462

Table 10: Results - Phase 1 - Manual Extract Mean Reciprocal Rank Summary

Equipment Name	Mean Reciprocal Rank
A10 Networks AX 3400	-
ADC FVM-19x700	0.049953
Agilent Spectrum Analyzer E4411B	0.056944
ATX MN1-16	0.070719
CableServ CHAS	0.030086
CableServ CHAS 2	0.078125
Casa Systems C100G	0.722222
Cisco 4506	0.85625
Cisco D9500	0.083333
Cisco UBR-RFSW-3X10 RF Switch	0.875
Cisco uBR10000	0.030792
Colomachine CM61	0.320455
Electroline TPS MS-100	0.063642
Electroline TPS SL-100	0.048423
Electroline TPS-32	0.07906
Generic 6-port AC Power Outlet	0.538462
JDSU Stealth Sweep Transceiver	0.066667
Juniper MX960	0.163175
Juniper SRX5600	1.000000
Motorola Apex-1000	0.865278
Motorola ARPD 1000	0.047687
Motorola NE2500	0.083333
Motorola Receiver	-
Motorola SE-1010	0.050228
Motorola SEM V8	1.000000
Motorola SEM V8 Cover	0.054505
Nortel OPTera Metro 5200	0.037037
Palmorex PMX 1500	0.034898
PCI Technologies MN2	0.27366
PCI Technologies MN5T	0.048299
PCI Technologies RMS PCI-81L 8-way Splitter	1.000000
PCI Technologies SCN-PP5-20F 5-20 Demark	0.079252
PCI Technologies TSG 4000R	0.090909
RGB Networks SEP48	0.047907
Scientific Atlanta D9510	-
Scientific Atlanta Prisma DTx	-
Standard TVM 550 II5	0.83333
Telect 10-10 GMT Fuse Panel	0.057799
Telect HPGMT15 Fuse Panel	0.061072
Telect KLM-GMT Fuse Panel	0.054750
Trilithic Super Series CT-2	0.083333

Table 11: Results - Phase 2 - Manual Extract Mean Reciprocal Rank Summary

Figure 27: Example of a plain faceplate, Kaveman 16 Digital V6

Figure 28: Example of a grill faceplate, Sun server

5.6 Full Rack Test Results

The full rack test aimed to test the quality of the entire prototype system. Rack images were input into the system, and then pieces of equipment were automatically isolated and tested against the equipment library. It is expected that if the edge detection methods performed equal to the manual cropping that the results would be similar to the previous extract tests. On that measure, we see mixed results. The phase one images overall rank better than those in the extract tests. The phase two images overall rank slightly worse than those in the extract tests.

There were a number of recurring factors that appeared to influence the results. The angle of the picture of the piece of equipment in the extract image compared to the template was a large factor. This is due to protruding switches and dials that appeared very differently from different directions. Furthermore, the position of the piece of equipment in the rack would also have an effect. A piece of equipment placed at the bottom of the rack would appear differently than one placed at the top, due to the angle between the piece of equipment and the camera.

The lighting on the piece of equipment had a strong impact on the ranking of the correct template. Although the aim was to mitigate this factor by ignoring the saturation from the HSV colour space, the impact was not completely eliminated. This factor was especially apparent in images that were "washed out" by bright fluorescent lights.

Factors that had a positive impact on getting a correct match were intrinsic to the piece of equipment being analyzed. Two factors appeared to have the most

Figure 29: Sun server with a colourful faceplate

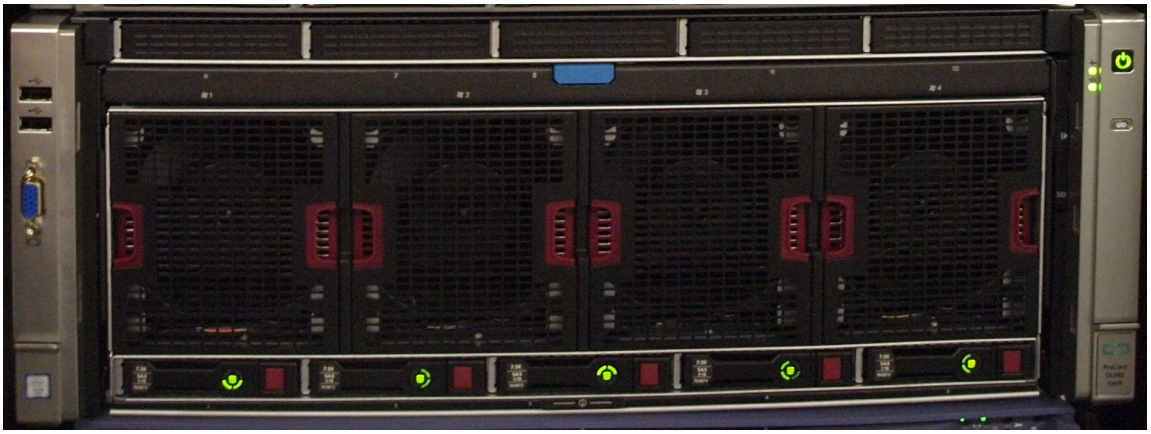

Figure 30: Example of a unique faceplate, HP ProLiant

impact. The first factor was a distinctive colour. Most pieces of equipment are varying shades of grey and black. Equipment that had a bright colour, such as the red Motorola Apex 100 or the mostly blue Cisco UBR-RFSW-3X10 RF Switch scored particularly well. Figure 29 shows an example of a colourful faceplate. The second factor is shape uniqueness. The majority of equipment have a grill, mesh, or plain faceplate. Examples of this are shown in Figures 27 and 28. Some equipment have a unique shape and performed very well. Figure 30 shows an example of a more unique faceplate.

Notably, template images taken with a mesh door in front of them, as shown in Figure 31, scored highly for almost all images being analyzed. It appears that the mesh adds to the number and variety of features detected in the image and results in a greater number of matches with features from other pieces of equipment.

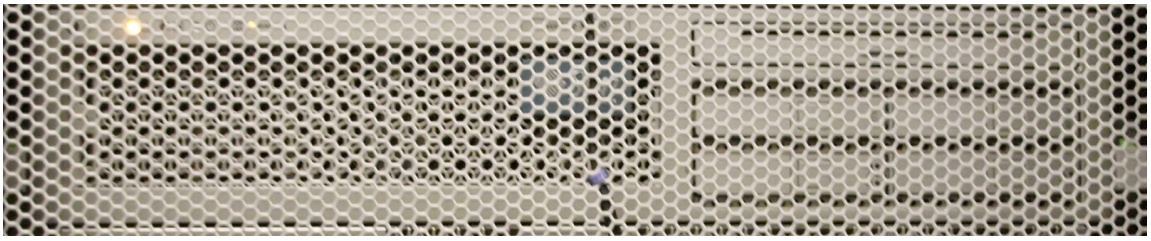

Figure 31: Example of a unique faceplate behind a mesh door

Equipment Name	**Number Correct Template Ranked**	**Average Rank**
ATX MN1	0	-
Cisco RFGW2	14	2.357143
Drake VM2410A Modulator	1	29
Jerrold S450M	1	26
Leitch 6800	1	12
Motorola Apex 1000	8	5.75
Motorola NE2500	10	7
Motorola SEM V8	1	11
MPEG2 DSR-4400	5	5
Telect GMT 10-10 Fuse Panel	2	8.5
Telect GMT 10-10 Fuse Panel 2	2	1
Telect HPGMT15 Fuse Panel	1	19

Table 12: Results - Phase 1 - Full System Test Summary

	Number of Images	**Percent of Total**
Total Equipment	51	-
Total Ranked	46	90.20%
Top 10	44	86.27%
Top 5	30	58.82%
Top 1	14	27.45%

Table 13: Results - Phase 1 - Full System High Ranking Summary

Equipment Name	Number Correct Template Ranked	Average Rank
ADC FVM-19x700	12	27.08333333
Agilent Spectrum Analyzer E4411B	4	25
ATX MN1-16	7	29.14285714
Aten KVM Switch	1	66
Aurora PF3000N-FM-00	1	1
CableServ CHAS	13	23.07692308
CableServ CHAS 2	9	33.33333333
Casa Systems C100G	5	2
Cisco D9500	2	27.5
Cisco UBR-RFSW-3X10 RF Switch	4	2.25
Cisco uBR10000	2	21.5
Colomachine CM61	3	17.33333333
Generic 6-port AC Power Outlet	4	8.5
JDSU Stealth Sweep Transceiver	1	32
Juniper MX960	9	3.444444444
Juniper SRX5600	3	7.666666667
Motorola Apex-1000	25	8.08
Motorola ARPD 1000	30	48.56666667
Motorola NE2500	21	25.0952381
Motorola SE-1010	26	30.23076923
Motorola SEM V8	1	15
Motorola SEM V8 Cover	9	29.11111111
Nortel OPTera Metro 5200	2	43
Palmorex PMX 1500	13	46.84615385
PCI Technologies MN2	8	15.125
PCI Technologies MN5T	68	32.72058824
PCI Technologies SCN-PP5-20F 5-20 Demark	8	30.875
PCI Technologies TSG 4000R	1	57
RGB Networks SEP48	4	42.5
Scientific Atlanta Continuum Modulator	6	26
Standard TVM 550 II5	1	22
Telect 10-10 GMT Fuse Panel	19	36.10526316
Telect HPGMT15 Fuse Panel	17	37.47058824
Telect KLM-GMT Fuse Panel	23	37.39130435
Telect TPA-GMT Fuse Panel	1	22
Terayon CP 7220	2	31.5
Trilithic Super Series CT-2	1	26
Ziptel Juniper MX480	2	1
Ziptel Routerboard MikroTik RB2011UiAS	1	22

Table 14: Results - Phase 2 - Full System Test Summary

	Number of Images	Percent of Total
Total Images	406	-
Total Ranked	369	90.89%
Top 10	43	11.65%
Top 5	36	9.76%
Top 1	23	6.23%

Table 15: Results - Phase 2 - Full System High Ranking Summary

Figures 16 and 17 show a summary of how highly the correct template was ranked. When compared to the extract test summaries (Figures 8 and 9) we see two different results for the different phases. In phase one the full rack performance was better than that of the extract tests, where 86.27% of images had the correct template ranked within the top 10, compared to 47.56% of images in the extract test. However, for phase two, we see that the full rack performance was worse, with only 11.65% of images ranking in the top ten versus the extract test result of 20.25%. The greater number of image templates in phase two likely played a part. However, the automated extract process likely had the largest impact. The process, as described in section 5.5.2, was initially designed for the images from phase one. By focusing on straight lines, this process was more precise than a human doing the manual cropping. However, the process typically performed slightly worse than a human when attempting to detect the edges of a piece of equipment for phase two images. This is due to the distortion from the fish-eye lens that remains even after calibration.

Another factor in the results for the phase two dataset is that with a larger set of server racks, there were a larger variation in the angles, shadows, occlusions and lighting conditions compared to the template images that were chosen. The prototype system has difficulty matching images that are very different, even if they are the same piece of equipment. For example, a template image with a lot of glare will always have difficulty matching a heavily shadowed or occluded image from the server rack.

5.6.1 Basic Statistical Performance

We see the same trend from the ranking summaries in the basic statistics in Figures 16 and 17. Figure 16 shows a decrease in all values when compared to the extract test values in Figure 16. Figure 17 shows a large increase in all values compared to the extract test values in Figure 16.

Mean	5.686275
Median	5
Standard Deviation	6.143257

Table 16: Results - Phase 1 - Full System Basic Statistics

Mean	29.39
Median	27.00
Standard Deviation	15.13

Table 17: Results - Phase 2 - Full System Basic Statistics

5.6.2 Mean Reciprocal Rank Performance

The mean reciprocal rank values show another view of the performance of the prototype system. The trends here match those seen in the previous sections. In the phase one values in Figure 18 there are a couple of high values (above 0.5). Typically this has occurred where very few instances of the piece of equipment existed. Those that did exist, matched well.

In the phase two values in Figure 19 there are many values in the range of 0.02-0.04. These values reflect the generally low ranking of the correct matches for this set of images.

Equipment Name	**Mean Reciprocal Rank**
Cisco RFGW2	0.633333333
Drake VM2410A Modulator	0.034482759
Jerrold S450M	0.038461538
Leitch 6800	0.083333333
Motorola Apex 1000	0.219047619
Motorola NE2500	0.190793651
Motorola SEM V8	0.090909091
MPEG2 DSR-4400	0.342222222
Telect GMT 10-10 Fuse Panel	0.202380952
Telect GMT 10-10 Fuse Panel 2	1
Telect HPGMT15 Fuse Panel	0.052631579

Table 18: Results - Phase 1 - Full System Mean Reciprocal Rank Summary

Equipment Name	Mean Reciprocal Rank
ADC FVM-19x700	0.037562511
Agilent Spectrum Analyzer E4411B	0.040918826
ATX MN1-16	0.039400859
Aten KVM Switch	0.015151515
Aurora PF3000N-FM-00	1
CableServ CHAS	0.043885855
CableServ CHAS 2	0.033078898
Casa Systems C100G	0.833333333
Cisco D9500	0.036472149
Cisco UBR-RFSW-3X10 RF Switch	0.5625
Cisco uBR10000	0.046536797
Colomachine CM61	0.061194653
Generic 6-port AC Power Outlet	0.347355769
JDSU Stealth Sweep Transceiver	0.03125
Juniper MX960	0.669360269
Juniper SRX5600	0.396011396
Motorola Apex-1000	0.416426384
Motorola ARPD 1000	0.022383703
Motorola NE2500	0.041403362
Motorola SE-1010	0.037316695
Motorola SEM V8	0.066666667
Motorola SEM V8 Cover	0.035236951
Nortel OPTera Metro 5200	0.028897849
Palmorex PMX 1500	0.022857829
PCI Technologies MN2	0.184965254
PCI Technologies MN5T	0.032891317
PCI Technologies SCN-PP5-20F 5-20 Demark	0.03432145
PCI Technologies TSG 4000R	0.01754386
RGB Networks SEP48	0.025999003
Scientific Atlanta Continuum Modulator	0.038575838
Standard TVM 550 II5	0.045454545
Telect 10-10 GMT Fuse Panel	0.02965893
Telect HPGMT15 Fuse Panel	0.028512083
Telect KLM-GMT Fuse Panel	0.027506311
Terayon CP 7220	0.031754032
Trilithic Super Series CT-2	0.038461538
Ziptel Juniper MX480	1
Ziptel Routerboard MikroTik RB2011UiAS	0.045454545

Table 19: Results - Phase 2 - Full System Mean Reciprocal Rank Summary

5.7 Time Performance

The prototype system was run on the system described in Section 5.2.1. For a rack image from phase one with a template library of 47 images, the system took approximately 50 minutes. For a rack image from phase two with a template library of 74 images, the system took approximately 80 minutes. The increase for phase two images is due to both the increase in the number of images in the template library, and the increase in size of the rack images.

6 Summary and Conclusion

6.1 Conclusions

This thesis proposes a system for image matching in a structured environment, such as a warehouse of products, where an insufficient number of training images are available for machine learning techniques. The proposed system works by first defining a library of template images, that is, a single image of a product that would be located in the warehouse. A data model is created for each piece of equipment to note the height, width, feature areas that are important, such as a logo or model number, and areas that are ignorable, such as a sticker or label that would not consistently appear on the product. Images taken of the warehouse shelves are also labelled to specify the height and width of the shelf being photographed. This information is used to assist the image matching process. For example, a product that is 23 inches wide cannot fit on a 19 inch wide shelf.

The image matching process involves two stages, edge detection and region matching. The edge detection stage proposes algorithms to analyze the images for the shelves on which the products are placed. Once the edges have been detected, the individual products are extracted from the image for further analysis. In the second stage, region matching takes place between the extracted images and the predefined library of template images. Based on the data model of the template image being compared, the images are dividing into sub-images, with the ignorable areas removed. The sub-images are compared based on shape features and colour clustering profiles, and assigned scores based on how similar they are. The feature and colour comparisons are repeated for each of the key feature regions defined in the template data model. The scores are then ranked to provide the user with a list of likely matches for each piece of equipment.

This approach was tested on two datasets. The datasets consisted of images of server racks from a data centre, and template images of equipment that was located in the racks. The first dataset was small and obtained under ideal conditions. The dataset also involved manual processing to create a full field of view of the equipment racks. The template images for the first dataset were obtained from the Internet, product manuals, and images of the equipment in situ. The second dataset was larger and obtained with a single picture using a large field of view camera. The template images for the second dataset were all obtained in situ, from the images of the server racks. Images from the second dataset required calibration before being processed by the rest of the system.

For the limited dataset, this system produces reasonable results, where 86.27%

of equipment have the correct template ranked in the top ten. However, the results were poor for a larger dataset with more variation in lighting, camera angle, and lens distortion from a wide angle, fish-eye lens. The larger dataset only ranked the correct template in the top ten 11.65% of the time. While the first dataset shows that the system can be useful under ideal conditions, the second dataset shows the limitations of the solution when applied to real world conditions. The median ranking of the correct template was 27 on the second dataset when compared to a median of 5 on the first dataset. With these results, the correct match could be expected to be on the "third page" of a ten results per page interface. Given that user expectations have been set for a correct match on the first page, this shows that the solution is not suitable as is to meet image recognition requirements. A larger template dataset, with multiple angles and lighting conditions could be explored as a possible solution in the future. However, a larger template library has a direct impact on runtime.

6.2 Future Work

This solution was devised for a specific use case, identifying objects within a warehouse (or a similar environment). The algorithms are therefore tailored to images that are taken at a direct angle, with the desired area for analysis in the centre of the image. This system would not work as designed if the photos could be taken at various angles. However, with changes it is possible this could be accommodated by adding multiple templates for a particular piece of equipment.

The results of the larger dataset show that the system does not deliver high quality results without ideal conditions. Variations in lighting and camera distortion are believed to have played a part in the poor results.

There are a number of potential avenues for future improvement to the described system. The most prominent among them would be the usage of machine learning to learn from the analysis, and feedback of the human operator over time to improve the results. Additionally, the existing system used only two attributes to compare a template image to an input image, shape and colour. Other attributes, such as texture and depth, provide additional dimensions upon which the images may be compared.

More targeted areas for improvement also exist. For example, the usage of multiple templates for a single piece of equipment to account for the way pieces of equipment appear different at various angles. Adding detection of occlusions such as stickers, and cables to automatically ignore them in comparisons may also improve matching scores. Attempting to limit the impact of lighting by levelling saturation, brightness, and contrast of the rack images may further improve results. Finally, al-

gorithm optimization and the usage of GPU's for computation could further improve processing time.

7 Bibliography

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Ali and D. Clausi. Using the Canny edge detector for feature extraction and\nenhancement of remote sensing images. In *IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No.01CH37217)*, volume 5, pages 2298–2300, 2001.

[3] D H Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognntton*, 13(2):111–122, 1981.

[4] J Bernal, F Vilarino, and J Sanchez. Feature Detectors and Feature Descriptors: Where We Are Now. Technical report, Universitat Autonoma de Barcelona, 2010.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features. *European Conference on Computer Vision (ECCV)*, pages 778–792, 2010.

[7] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[8] Nick Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, Boston, MA, 2009.

[9] Mei Fang, Gx Yue, and Qc Yu. The study on an application of otsu method in canny operator. *International Symposium on Information ...*, 2(4):109–112, 2009.

[10] Theo Gevers and Arnold W M Smeulders. PicToSeek: combining color and shape invariant features for image retrieval. *IEEE Transactions on Image Processing*, 9(1):102–119, 2000.

[11] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Clustering Validity Methods: Part {II}. *ACM SIGMOD Record*, 31(3):19–27, 2002.

[12] B. R. Heap. Permutations by Interchanges. *The Computer Journal*, 6(3):293–298, 1963.

[13] Qingming Huang, Wen Gao, and Wenjian Cai. Thresholding technique with adaptive window selection for uneven lighting image. *Pattern Recognition Letters*, 26:801–808, 2005.

[14] Itseez. *The OpenCV Reference Manual*, 3.2.0 edition, August 2017.

[15] Anil K Jain and Aditya Vailaya. Image Retrieval using Color and Shape. *Pattern Recognition*, 29:1233–1244, 1995.

[16] Ramesh Jain, Rangachar Kasturi, and Brian Schunck. Chapter 4: Image Filtering. In *Machine Vision*, pages 112–139. McGraw-Hill, New York, NY, 1995.

[17] Ramesh Jain, Rangachar Kasturi, and Brian Schunck. Edge detection. In *Machine Vision*, chapter 5, pages 140–185. McGraw-Hill, New York, NY, 1995.

[18] George H. Joblove and Donald Greenberg. Color Spaces for Computer Graphics. *ACM siggraph computer graphics*, 12(3):20–25, 1978.

[19] Mohan S. Kankanhalli, Babu M. Mehtre, and Jian Kang Wu. Cluster-based Color Matching for Image Retrieval. *Pattern Recognition*, 29(4):701–708, 1996.

[20] N. Kiryati, Y. Eldar, and A. M. Bruckstein. A Probabilistic Hough transform. *Pattern Recognition*, 24(4):303–316, 1991.

[21] Martin Krzywinski. Image color summarizer, 2006–2016. [Online; accessed December 20, 2016].

[22] Pawan Lingras, Farhana Haider, and Matt Triff. Granular meta-clustering based on hierarchical, network, and temporal connections. *Granular Computing*, 1(1):71–92, 2016.

[23] L Lucchese and Sanjit K Mitra. Unsupervised Segmentation of Color Images Based on k-means Clustering in the Chromaticity Plane. In *IEEE Workshop on Content based Access of Image and Video Libraries (CBAIVL'99)*, pages 74–78, 1999.

[24] J Macqueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1(233):281–297, 1967.

[25] J. Matas, C. Galambos, and J. Kittler. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding*, 78(1):119–137, 2000.

[26] Peter Meer, Doron Mintz, Azriel Rosenfeld, and Dong Yoon Kim. Robust Regression Methods for Computer Vision: A Review. *International Journal of Computer Vision*, 6(593):59–70, 1991.

[27] Babu M. Mehtre, Mohan S. Kankanhalli, A. Desai Narasimhalu, and Guo Chang Man. Color Matching for Image Retrieval. *Pattern Recognition Letters*, 16(3):325–331, 1995.

[28] Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(10):1615–1630, 2005.

[29] Pandu Nayak. Introduction to information retrieval, 2013. [Online; accessed July 1, 2017].

[30] Hui-Fuang Ng and Y J Zhang. Automatic thresholding for defect detection. *Pattern Recognition Letters*, 2006.

[31] Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9(1):62–66, 1979.

[32] Leif Peterson. K-nearest neighbor, 2009.

[33] B.G Prasad, K.K Biswas, and S.K Gupta. Region-based image retrieval using integrated color, shape, and location index. *Computer Vision and Image Understanding*, 94(1-3):193–233, 2004.

[34] Yu Qiao, Qingmao Hu, Guoyu Qian, Suhuai Luo, and Wieslaw L Nowinski. Thresholding based on variance and intensity contrast. *Pattern Recognition*, 40:596–608, 2007.

[35] Shiv Ram Dubey and As Jalal. Robust Approach for Fruit and Vegetable Classification. *Procedia Engineering*, 38:3449–3453, 2012.

[36] Paul L Rosin. Measuring Corner Properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999.

[37] Edward Rosten and Tom Drummond. Machine Learning for High Speed Corner Detection. *Computer Vision – ECCV 2006*, 1:430–443, 2006.

[38] Peter Rousseeuw. Least Median of Squares Regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[39] Ethan Rublee and Gary Bradski. ORB - an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

[40] Mehmet Sezgin and Bulent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004.

[41] SharkD. Hsv color solid cylinder, 2015. [Online; accessed September 11, 2016].

[42] Alvy Ray Smith. Color gamut transform pairs. *ACM SIGGRAPH Computer Graphics*, 12(3):12–19, 1978.

[43] Stephen M. Stigler. Gauss and the Invention of Least Squares. *The Annals of Statistics*, 9(3):465–474, 1981.

[44] Michael J Swain and Dana H Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[45] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe Van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[46] Hetal J Vala and Astha Baxi. A Review on Otsu Image Segmentation Algorithm. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 2(2):387–389, 2013.

[47] Nuno Vasconcelos. Edges, interpolation, templates, 2009.

[48] Dengsheng Zhang and Guojun Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1–19, 2004.

[49] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

8 Appendices

8.1 Appendix 1: Phase 1 Template Library Equipment

The following template images were included in the phase one tests referenced in Chapter 5.

1. A10 AX3200-12_1RU_19in
2. Alcatel-Lucent 7750 SR c-12_5RU_19in
3. Apex 1000_1RU_19in
4. ATX MN1_1RU_19in
5. ATX MN5_5RU_19in
6. ATX SCN-RG7_1RU_19in
7. Aurora CH3000N_3RU_19in
8. Canare Demark_1RU_19in
9. Ciena CMD44_2RU_19in
10. Cienna OME6500 32-slot_22RU_23in
11. Cisco 2950-24_1U_19in
12. Cisco 3845_3RU_19in
13. Cisco 3945_3RU_19in
14. Cisco 4506 AC_10RU_19in
15. Cisco 4506-E_10RU_19in
16. Cisco 6509_14RU_19in
17. Cisco 7609_21RU_19in
18. Cisco CRS-1 8-Slot_22RU_19in
19. Cisco CRS-16_48RU_21in

20. Cisco RFGW2_1RU_19in
21. Drake VM2410A Modulator_1RU_19in
22. Electroline TPS-32_1RU_19in
23. Electroline TPS-MS-100_1RU_19in
24. Electroline TPS-SL-100_1RU_19in
25. Fujitsu FW7500_13RU_23in
26. Harmonic NSG 9000 Eqam_2RU_19in
27. Jerrold S450M_1RU_19in
28. Juniper EX4550_1RU_19in
29. Juniper MX960_21RU_19in
30. Leitch 6800_2RU_19in
31. Motorola ARPD 1000_1RU_19in
32. Motorola NE2500_1RU_19in
33. Motorola OM2000_1RU_19in
34. Motorola SEM V8_1RU_19in
35. MPEG2 DSR-4400_2RU-19in
36. Nortel NT0H31AH_1RU_19in
37. Nortel NT0H32AF_1RU_19in
38. Nortel NT0H32AH_1RU_19in
39. Nortel NT0H32BF_1RU_19in
40. Nortel Optera Metro 5200_11RU_19in w-Cover
41. Nortel Optera Metro 5200_11RU_19in
42. PCI SCN-RG5_1RU_19in

43. RGB SEP48_1RU_19in
44. Telect GMT 10-10 2_1RU_19in
45. Telect GMT 10-10 Fuse Panel_1RU_19in
46. Telect HPGMT15 Fuse Panel_1RU_19in
47. Telect KLM-GMT 4-4 Fuse Panel_1RU_19in

8.2 Appendix 2: Phase 2 Template Library Equipment

The following template images were included in the phase two tests referenced in Chapter 5.

1. A10 Networks AX 3400_19in_1RU
2. A10 Networks AX3200-12_19in_1RU
3. A10 Networks AX3200-12_19in_1RU_1
4. ADC FVM-19x700_23in_4RU
5. Agilent Spectrum Analyzer E4411B_3RU_19in
6. Alcatel-Lucent 7360-ISAM-FX4_02_19in_5RU
7. Alcatel-Lucent 7750-SRc-12_19in_5RU
8. Aten KVM Switch_19in_1RU
9. ATX MN1-16_19in_1RU_1
10. ATX MN1-16_19in_1RU_2
11. ATX MN1-16_19in_1RU_3
12. Aurora PF3000N-FM-00_19in_4RU
13. CableServ CHAS-2_19in_3RU_07
14. CableServ CHAS_3RU_19in
15. CableServ CHAS_3RU_19in_2
16. Casa Systems C100G_13RU_19in
17. Cisco 3945_19in_3RU
18. Cisco 4506_19in_10RU
19. Cisco 4506_19in_1RU
20. Cisco D9500_23in_1RU

21. Cisco UBR-RFSW-3X10 RF Switch_3RU_19in
22. Cisco uBR10000_18RU_19in
23. Colomachine CM61_1RU_19in
24. Electroline TPS MS-100_1RU_19in
25. Electroline TPS SL-100_1RU_19in
26. Electroline TPS-32_1RU_19in
27. Electroline TPS-SL-100_1RU_19in
28. Generic 6-port AC Power Outlet_2RU_19in
29. Generic Shelf_19in_2RU_05
30. JDSU Stealth Sweep Transceiver_2RU_19in
31. Juniper MX960_19in_21RU
32. Juniper MX960_19in_21RU_1
33. Juniper MX960_19in_21RU_O
34. Juniper SRX5600_19in_8RU
35. Juniper SRX5600_19in_8RU_O
36. Motorola Apex-1000_1RU_19in
37. Motorola ARPD 1000_1RU_19in
38. Motorola NE2500_23in_1RU
39. Motorola SE-1010_23in_1RU
40. Motorola SEM V8 Cover_1RU_19in
41. Motorola SEM V8_1RU_19in
42. Nortel OPTera Metro 3500_19in_10RU
43. Nortel OPTera Metro 5200_19in_11RU

44. Omnitron 8205-2_19in_2RU
45. Palmorex PMX 1500_23in_1RU
46. PCI Technologies MN2_1RU_19in
47. PCI Technologies MN5T_5RU_19in
48. PCI Technologies RMS PCI-81L 8-way Splitter_1RU_19in
49. PCI Technologies SCN-PP5-20F 5-20 Demark_1RU_19in
50. PCI Technologies TSG 4000R_1RU_19in
51. Perftech MX Director 2000_19in_1RU
52. RGB Networks SEP48_1RU_19in
53. Scientific Atlanta Continuum Modulator _19in_5RU_02
54. Scientific Atlanta Prisma DTx 2540_19in_1RU
55. Standard TVM 550 II5_1RU_19in
56. Telect 10-10 GMT Fuse Panel_19in_1RU
57. Telect 10-10 GMT Fuse Panel_19in_1RU_01
58. Telect HPGMT15 Fuse Panel_1RU_19in
59. Telect KLM-GMT Fuse Panel_19in_1RU
60. Telect KLM-GMT Fuse Panel_19in_1RU_01
61. Telect KLM-GMT Fuse Panel_19in_1RU_02
62. Telect TPA-GMT Fuse Panel_19in_1RU
63. Terayon CP 7220_19in_1RU
64. Trilithic Super Series CT-2_1RU_19in
65. Ziptel AudioCodes Mediant-3000_19in_2RU
66. Ziptel AudioCodes Mediant-3000_19in_2RU_O

67. Ziptel Digi Port Server TS16_19in_1RU
68. Ziptel Digi Port Server TS16_19in_1RU_O
69. Ziptel HP BLc7000_19in_10RU
70. Ziptel HP BLc7000_19in_10RU_O
71. Ziptel Juniper MX480_19in_8RU
72. Ziptel Juniper MX480_19in_8RU_O
73. Ziptel Routerboard MikroTik RB2011UiAS_19in_1RU
74. Ziptel Routerboard MikroTik RB2011UiAS_19in_1RU_O