ELEC4630 Assignment 1

Samuel Eadie - 44353607

March 25, 2018

1 Relevant Background Theory

1.1 Colour Spaces

There are many coordinate systems that can describe colour. Two of the most frequently used are the Red-Green-Blue (RGB) and Hue-Saturation-Value (HSV) coordinate systems. RGB uses the colours red, green and blue as orthogonal basis vectors in a cartesian coordinate system. In HSV, hue represents the perceived colour, saturation the colourfulness and value the brightness. HSV is a polar coordinate system that better models humans' perception of colour.

1.2 Thresholding

Thresholding is a simple quantisation method to segment images based on a property. Often, thresholding is performed on the grayscale of an image to binarize it. All pixels in an image that are above the threshold value for the given property are replaced with white, otherwise they are turned black. Thresholding can be performed globally with a constant thresholding value, or locally where the thresholding value is dependant on the position of the pixel. The latter is generally useful in images with uncontrolled or uneven lighting (e.g. outdoors).

1.3 Morphological Transformations

Morphological transformations cover a set of operations based on shapes in an image. In these transformations, each pixel is assigned a value depending on the pixels in the neighbourhood of the former. The neighbourhood is defined by a structuring element which characterises the shape the transformation relates to.

- **Dilation:** A pixel is set if any of the pixels in its neighbourhood are set. This results in an expansion of objects in the shape of the structuring element.
- **Erosion:** A pixel is set if all of the pixels in its neighbourhood are set. This results in a shrinking of objects in the shape of the structuring element.
- **Closing:** A dilation followed by an erosion is performed with a common structuring element. This removes holes that are a similar shape to the structuring element from objects.
- **Opening:** An erosion followed by a dilation is performed with a common structuring element. This expands holes and removes noise that are a similar shape to the structuring element from objects.
- **Gradient:** The difference between the dilation of an image and the erosion of an image with a common structuring element. This outlines objects.
- **Skeletonisation:** A process for reducing components to a pixel wide skeleton that ideally preserves the extent and connectivity of the component

1.4 Cross Correlation

A cross correlation provides a measure of the similarity between two signals. It is computationally similar to a convolution but does not rotate either signals prior to operation. A two-dimensional cross correlation is useful in image processing for locating a template in an image.

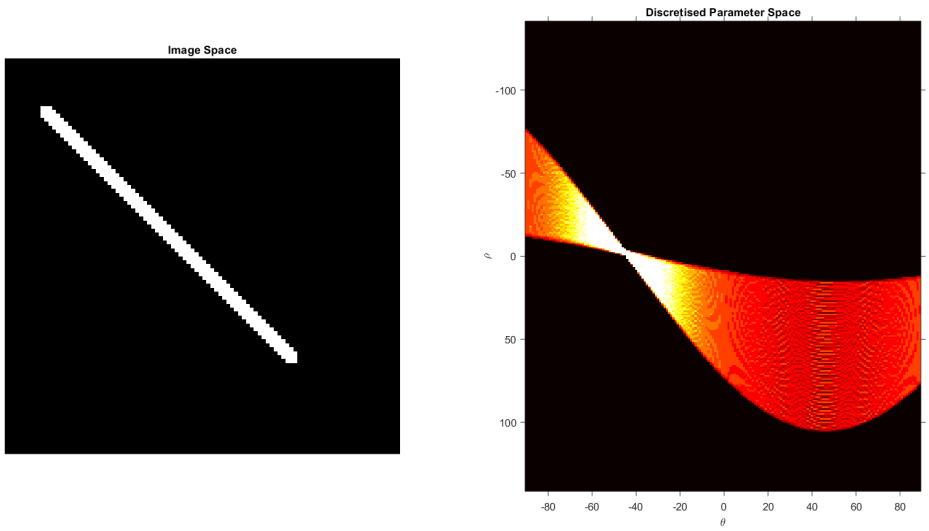

Figure 1: Hough Transform of a Line to Parameter Space

1.5 Hough Transformation

The Hough Transform is a feature extraction technique well-known for identifying lines in images. That said, the Hough transform can also identify other shapes. The transform maps from image space to a discretised parameter space using a type of voting scheme. Once transformed, the maximum intensity points in the parameter space most likely correspond to the parameter shape in image space.

For line detection, a polar line representation is used for the parameter space to avoid an infinite parameter domain caused by the Cartesian representation of vertical lines. In polar coordinates, a line is represented as $x\cos\theta + y\sin\theta = \rho$. Therefore, every point in image space contributes a sinusoid in this polar coordinate parameter space. If a line is present in the image, the sinusoidal contribution of each point on the line will intersect at a point of maximum intensity. This intersection point in parameter space corresponds to a line in image space. This is shown in Figure 1.

2 License Plate Segmentation

2.1 Introduction

Automatic object segmentation is a common pre-processing step in image processing applications. The segmentation of license plates in images is an important pre-processing step in automatic number plate reading applications. A license plate segmentation technique is required to outline the license plate in images. This system must be robust to successfully segment number plates of varying size, colour, orientation, lighting, context and formatting.

2.2 Method

The final approach centred around filtering objects out of the image based on their shape. As a preprocessing step, the image was binarised via grayscaling and thresholding. The process consists of eight stages.

1. Preprocessing is performed to binarise the image. This involves converting the image to grayscale before thresholding. The threshold was designed to clearly distinguish the number plate letters from the number plate.
2. The largest component, measured in number of pixels, was removed from the image.
3. Components smaller than a minimum threshold were removed. These were deemed too small to represent numbers and are irrelevant noise.
4. Components adjacent to a border were removed since the plates are generally relatively centred. Plates on the border are most likely incomplete and not useful.
5. Morphological transformations were performed to join the letters into a blob spanning the license plate.
6. The ratio between each components' maximum and minimum axis lengths were calculated. Components with a ratio outside the normal range for a number plate were removed.
7. The largest component was selected as most likely being the number plate.
8. The convex hull of the identified component was calculated. The convex hull was used to outline the number plate on the original image.

A typical image, at various stages throughout this processing, is shown in Figure 2.

2.3 Results

This approach successfully segmented 19 of the 26 test images. The successful segmentations are shown in Figure 3 and the unsuccessful segmentations are shown in Figure 4.

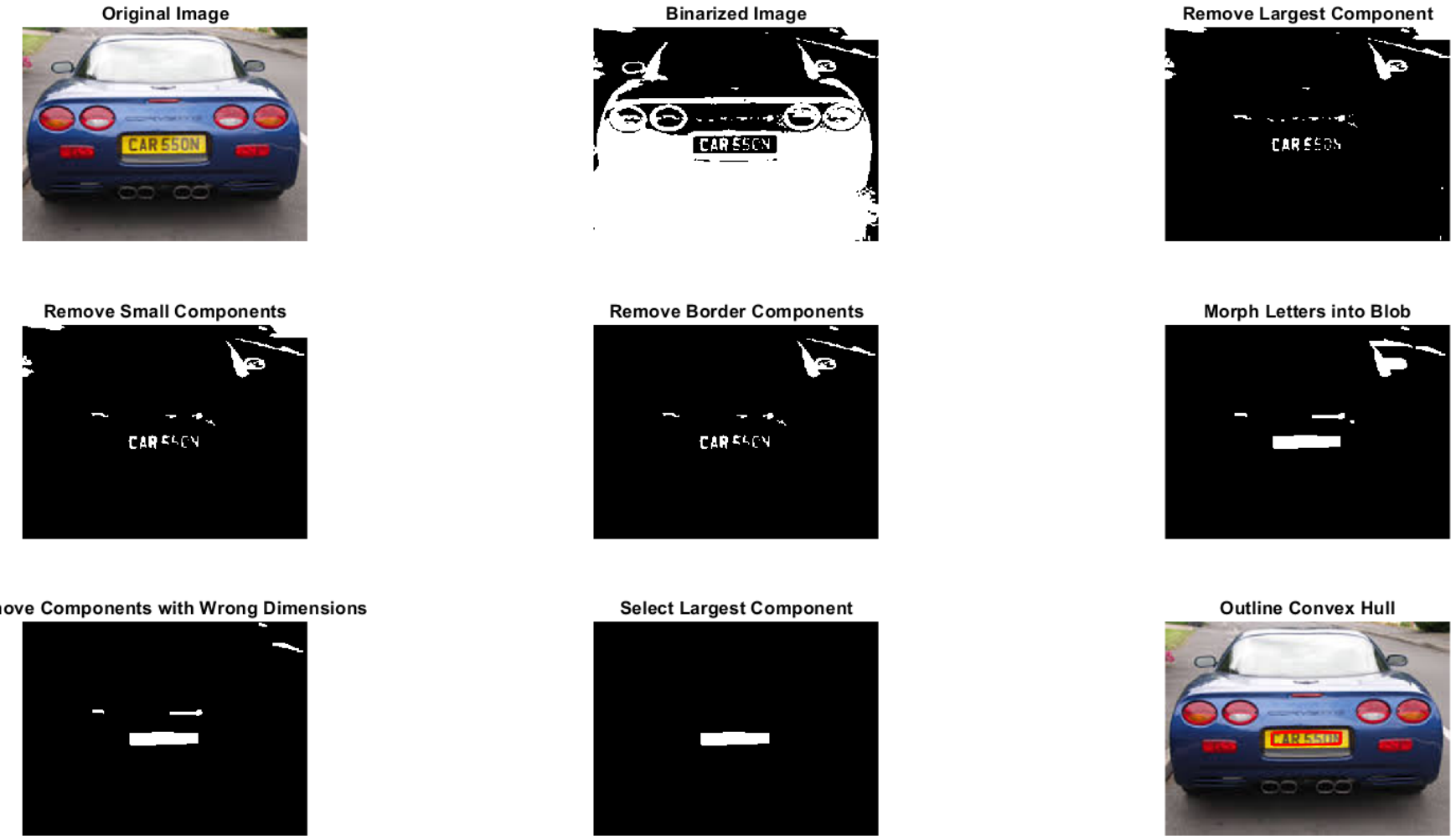

Figure 2: License Plate Segmentation Process

Figure 3: Successful License Plate Segmentations

Figure 4: Unsuccessful License Plate Segmentations

2.4 Discussion

While the segmentation approach was largely successful, it struggled on several number plates because of the difficulty in developing a robust pre-processing thresholding for plates of varying size, colour, orientation, lighting, context and formatting. All unsuccessful segmentations were preceded by a poor thresholding, as shown in Figure 5. This thresholding aimed to distinguish the number plate lettering from the plate. This contrast was deemed the most consistent characteristic across images and thus the most robust property to exploit for this segmentation.

2.4.1 Alternative Approaches

Alternative approaches centred around distinguishing the number plate from it's background. This approach struggled because of the inconsistencies across images in the differentiating domain; plates were generally distinguishable from their background in either intensity or hue domain, rarely both and sometimes neither (Figure 6).

Approaches based on the image's hue value were also hindered by the .jpg file format. The quantisation of JPEG images into blocks for Discrete Cosine Transform processing during JPEG compression results in block artefacts in the hue domain. A hue domain approach was tested but was largely unsuccessful. Examples of images un/successfully segmented with this approach are shown in Figure 7.

2.4.2 Possible Improvements

The final approach could possibly be improved by:

- **Letter & Number Recognition:** Instead of choosing the largest blob in the final stage, letter & number recognition could be used as a criterion for the final selection. That said, this license plate segmentation is most likely a preprocessing step in a number plate recognition problem.
- **Local Thresholding:** Currently, a global threshold is applied to the entire grayscale image in the pre-processing binarization. Instead, a local thresholding could be implemented. In such an approach, the pixel's threshold value is the average of it's neighbourhood. An example of this is shown in Figure 8.

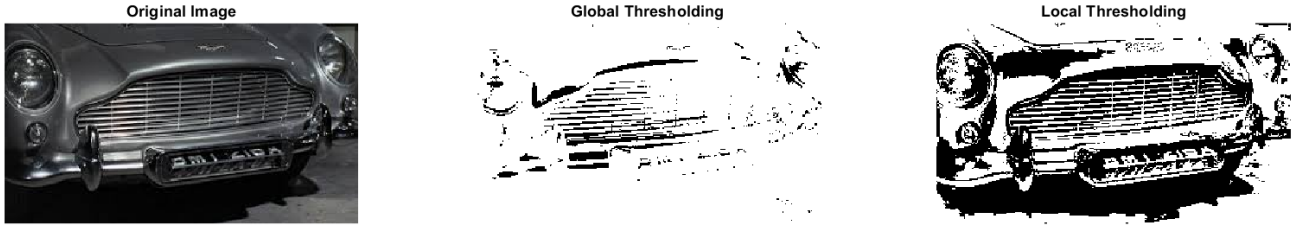

Figure 8: Local Thresholding Example

Figure 5: Poor Thresholding in Unsuccessful Segmentations

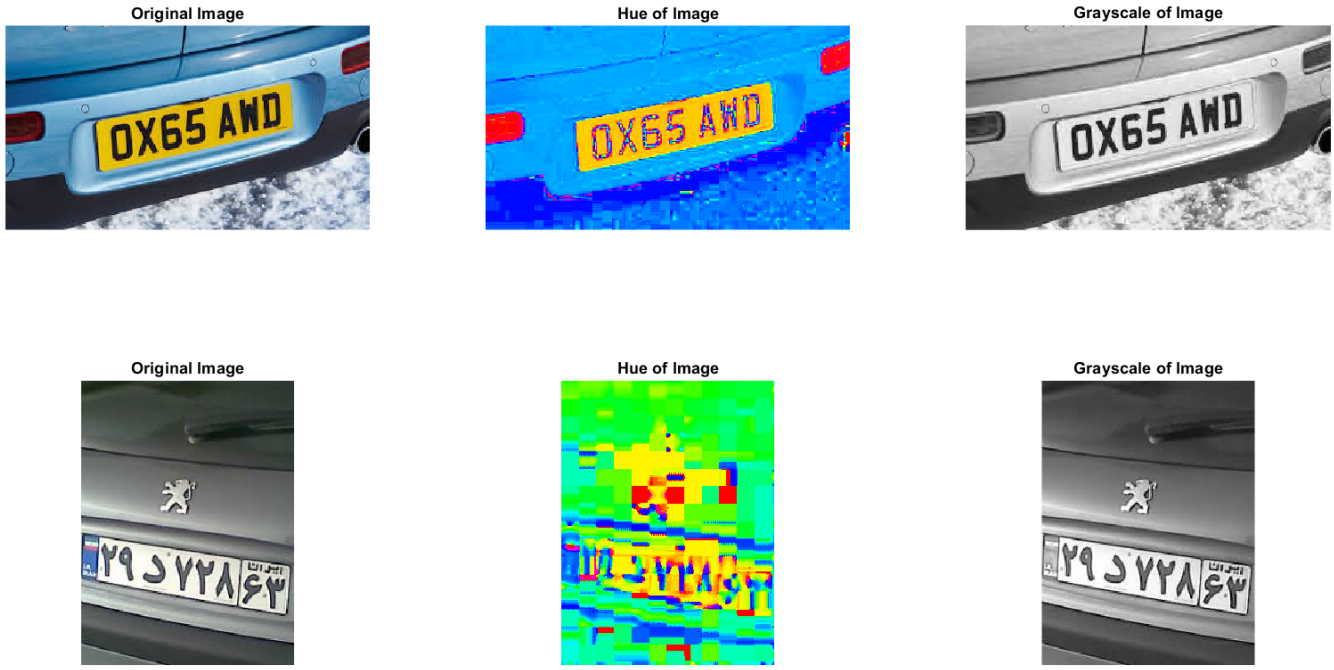

Figure 6: Examples of Images in Hue and Intensity Domains

Figure 7: Hue Domain Approach Example Segmentations

3 Pantograph Video Processing

3.1 Introduction

Electricity is provided to trains by an overhead power line which slides across a pantograph-supported carbon brush. It is important to track the position of the power line on this carbon brush to ensure there is a consistent and reliable power supply for the train.

As such, an automatic video processing system is required to track the intersection of the power line with the carbon brush throughout the train's operation. This system must distinguish the power line from the suspension cable which hangs above and is visible in the field of view.

3.2 Method

The overarching approach to this task is outlined in Figure 9. This process is repeated for each frame.

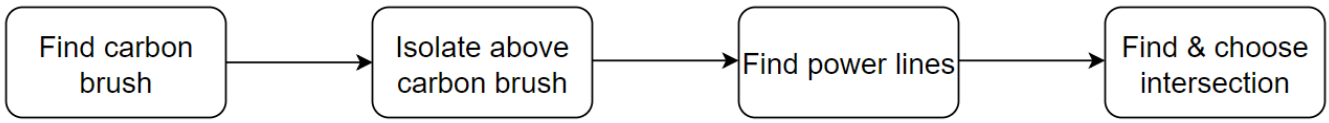

Figure 9: Method Overview

3.2.1 Carbon Brush Localisation

The process to locate the carbon brush consists of several stages.

1. The frame to be processed is read in
2. The frame is cropped to remove the superimposed metadata text at the bottom and the black border
3. The frame, after being converted to grayscale, is thresholded and binarized to isolate the pantograph, carbon brush and power lines
4. Morphological transformations are performed on the binarized image to fill the pantograph structure below the carbon brush. This involves a horizontal dilation, and a vertical dilation, close and erosion respectively. A template image was extracted from the first video frame at this stage in the process for cross correlation (Figure 10).
5. A cross correlation is performed between the morphologically-transformed carbon brush from the first video frame and the currently-processing frame to locate the pantograph.
6. The position of the carbon brush, relative to the template image, is known. Therefore, the position of the carbon brush can be found and plotted.

Figure 10: Morphologically-Transformed Template Image

A typical frame, at various stages throughout this processing, is shown in Figure 11.

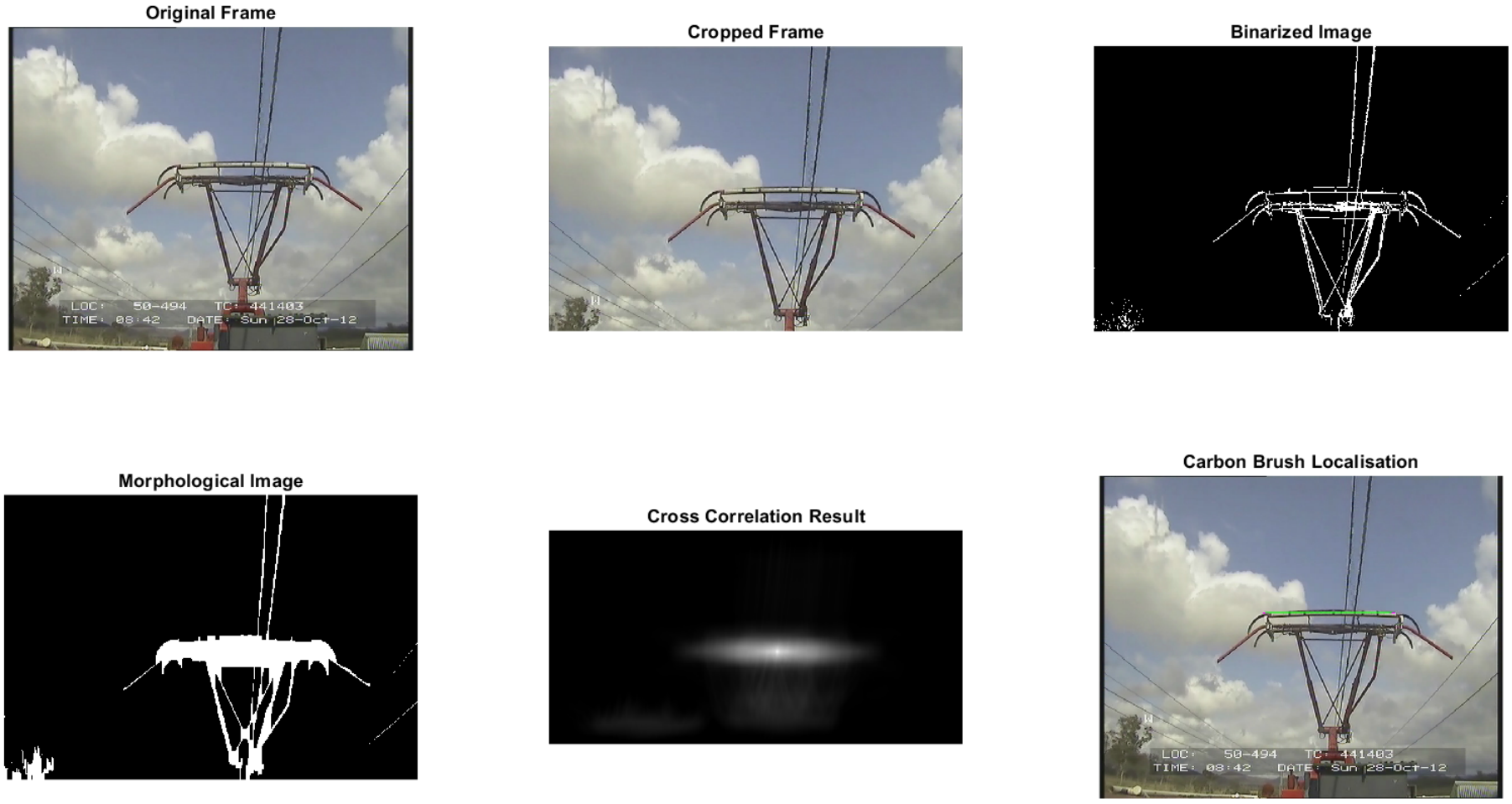

Figure 11: Carbon Brush Localisation Process

3.2.2 Power Lines

The power lines are located above the carbon brush; therefore, since the carbon brush has already been located, a new search area is outlined. This is shown in Figure 12.

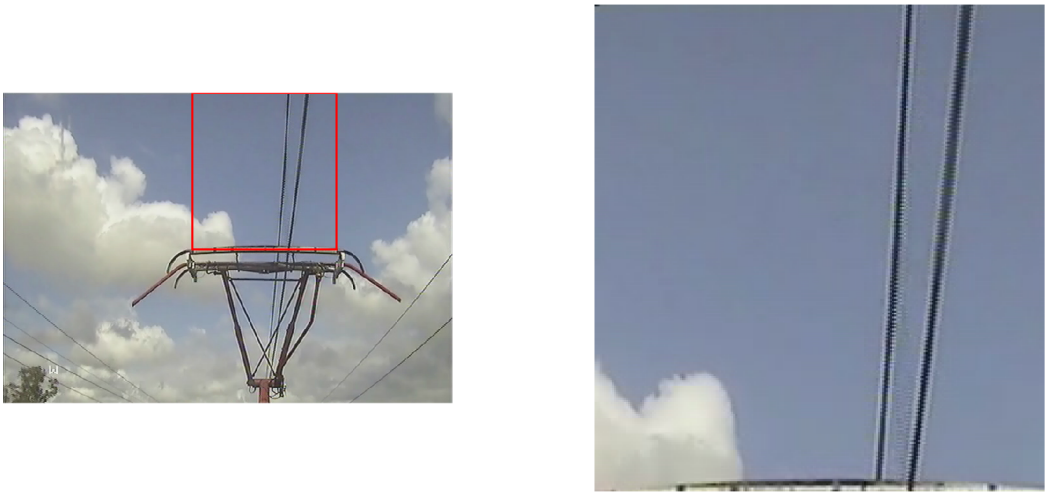

Figure 12: Power Line Search Section Outline

The Hough Transform is used to find prominent lines in the search section. The transform is applied on a skeletonised version of the pre-morphology binarized image used previously in the carbon brush localisation process. This skeletonisation thins the power lines to one pixel wide, increasing the accuracy of the Hough transform. This skeletonisation is shown in Figure 13.

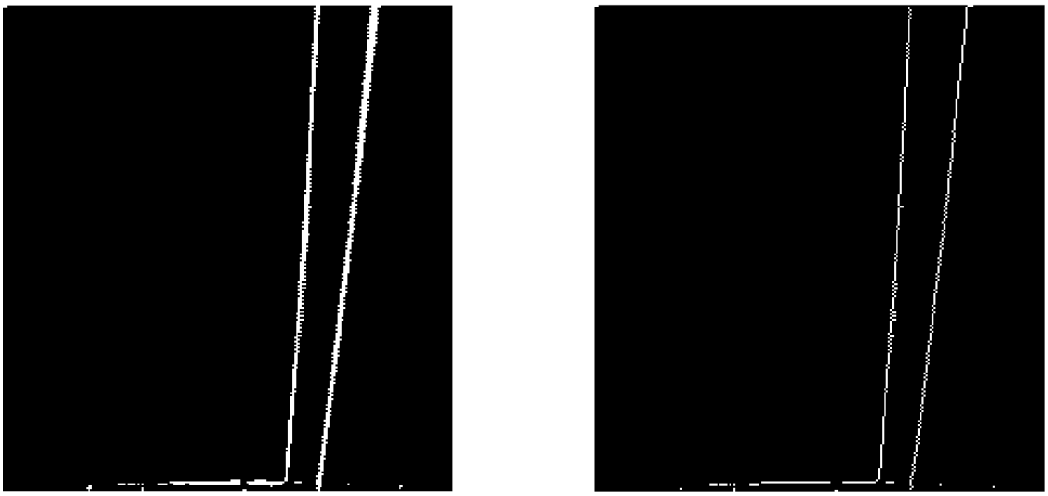

Figure 13: Skeletonisation of Binarized Image

The Hough Transform Matrix is sliced to extract only lines within a threshold deviation from vertical. From this, lines longer than 75% of the search section height are selected in decreasing

order of intensity. After each line selection, the neighbourhood around the selected line is zeroed to avoid choosing near-identical lines twice. This continues until no further lines have an intensity greater than half the maximum line intensity.

The power line can be distinguished from the support cable since it is always at a larger angle to the vertical than the latter. From this, the power line is selected from the lines generated through the Hough transform process.

Now, having lines for both the power line and carbon brush, their intersection can be calculated. The result is plotted and stored. The resulting image is shown in Figure 14.

3.2.3 Temporal Thresholding

The Pantograph video processor was implemented in an object-oriented approach. This allows the class to store state across the processing of consecutive frames. This was used to perform simple temporal thresholding under the key premise that objects can only move a certain distance between frames. When locating the carbon brush and hence the power line's intersection, a threshold distance around the point's previous location was deemed legal.

3.3 Results

The outlined approach was successful in locating, distinguishing and tracking the power line as it slid across the carbon brush. Figure 14 shows a typical video frame with the carbon brush and distinguished power line successfully outlined and their intersection found.

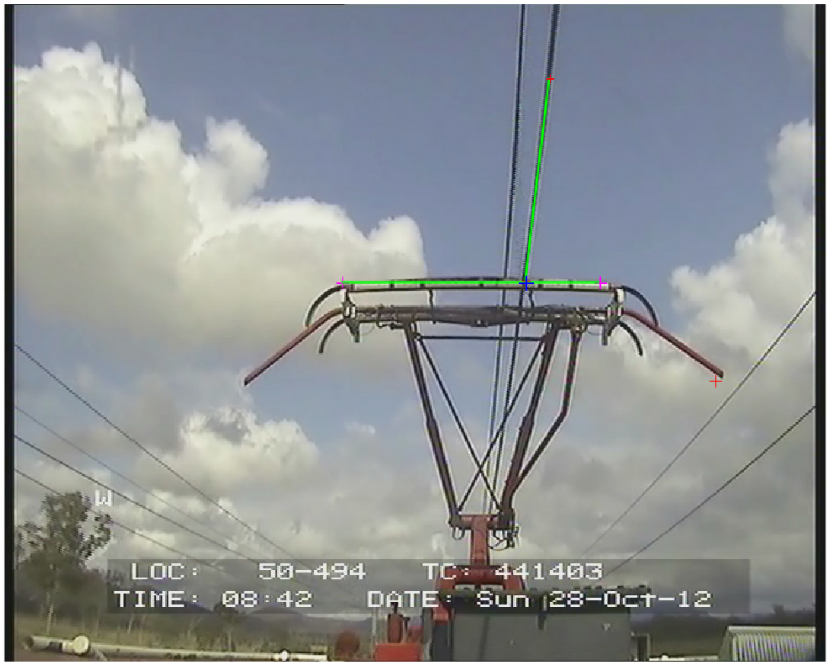

Figure 14: Pantograph labelled with Carbon Brush, Power Line and Intersection

The position of the power line's intersection with the carbon brush over time was plotted in Figure 15.

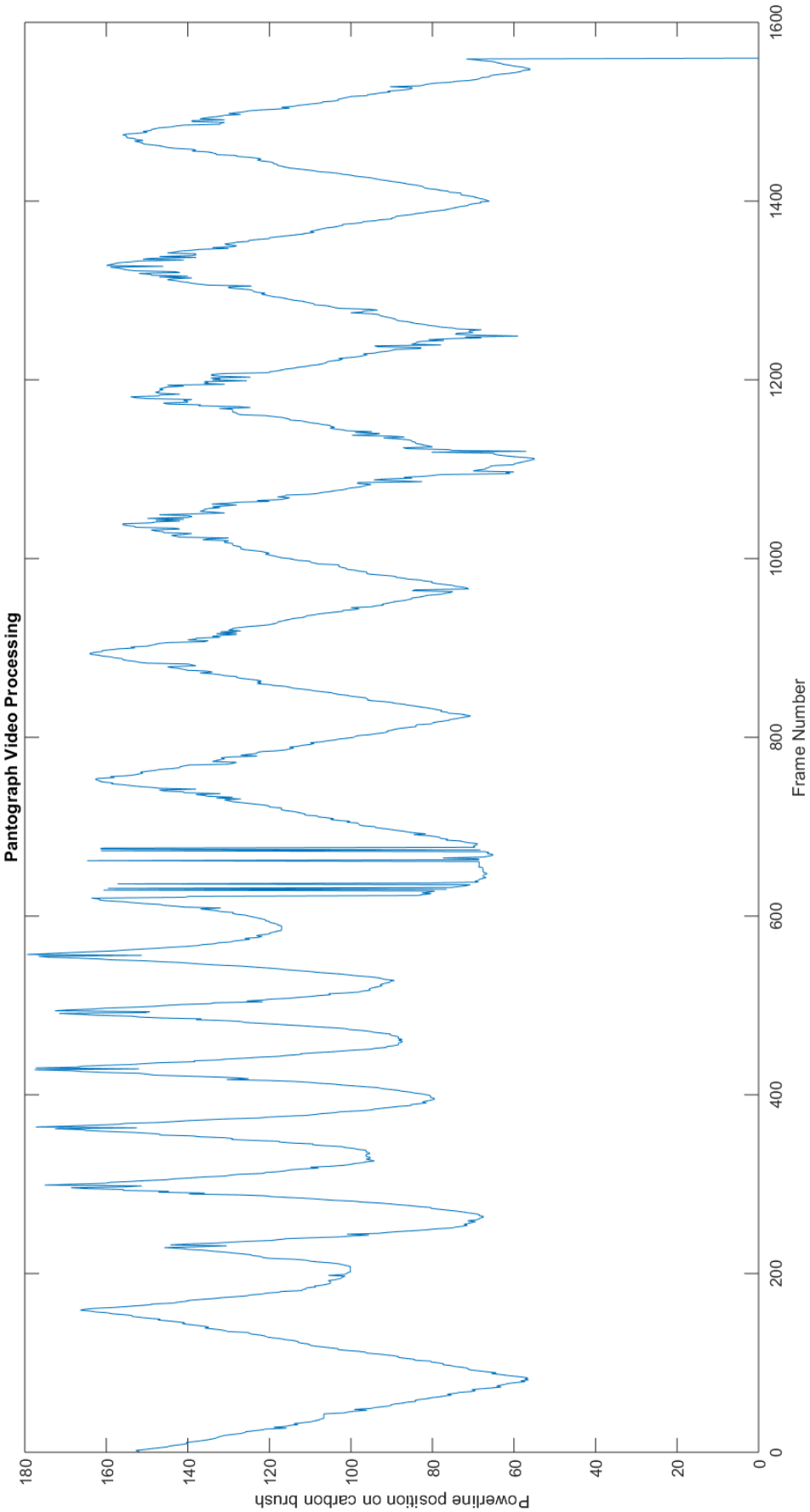

Figure 15: Power Line Position on Carbon Brush over time

3.4 Discussion

Several factors simplified the power line tracking problem.

- Constant camera position, angle and field of view.
- Relatively constant object position, size and orientation
- Significant contrast between objects of interest and background

While the design was largely successful in tracking the power line's intersection with the carbon brush, it struggled to smoothly transition to the new power line. Despite detecting and switching to the new power line, it struggled to uniformly stay with it, often momentarily jumping back to the original power line. Unlike the carbon brush, the power lines were unable to be temporally thresholded due to this switch; this thresholding would have prevented this oscillation. The transition period, during which the power lines were switched, is outlined in Figure 16.

3.4.1 Alternative Approaches

Several alternative approaches were considered for the binarization of frames as a preprocessing step to the Hough Transform. The binarized frames for these approaches are compared against the morphological preprocessing performed prior to carbon brush localisation in the final approach (Figure 17).

A Canny Edge Detector based approach to outline the pantograph, carbon brush and power line before Hough Transformation was trialled. This produced less connected lines and the multitude of horizontal lines around the carbon brush hindered the Hough Transform. A temporal approach was trialled; the difference in grayscale of consecutive frames was calculated and then thresholded. This technique was susceptible to movement in clouds behind the power lines. Additionally, when the power lines were relatively stationary, around their turning points for example, this technique could not outline them.

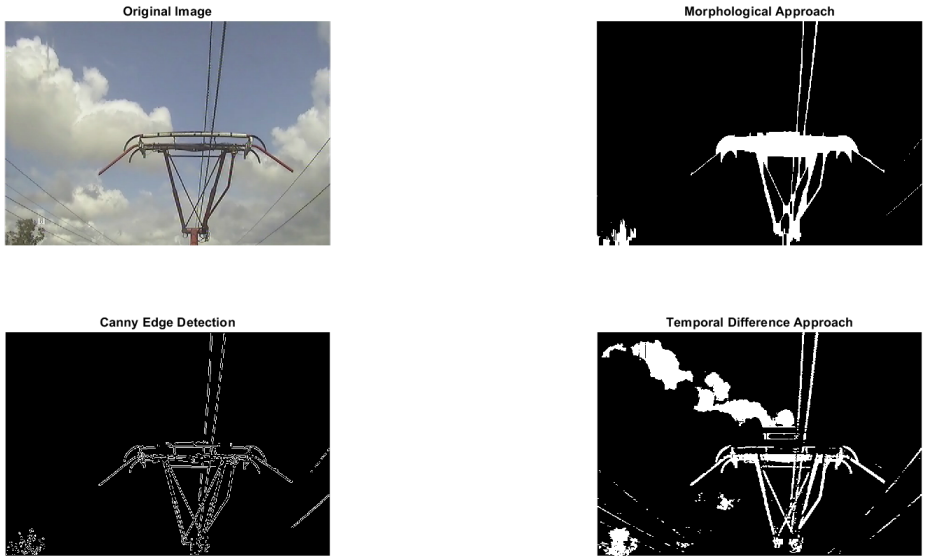

Figure 17: Alternative Approaches to Frame Binarization

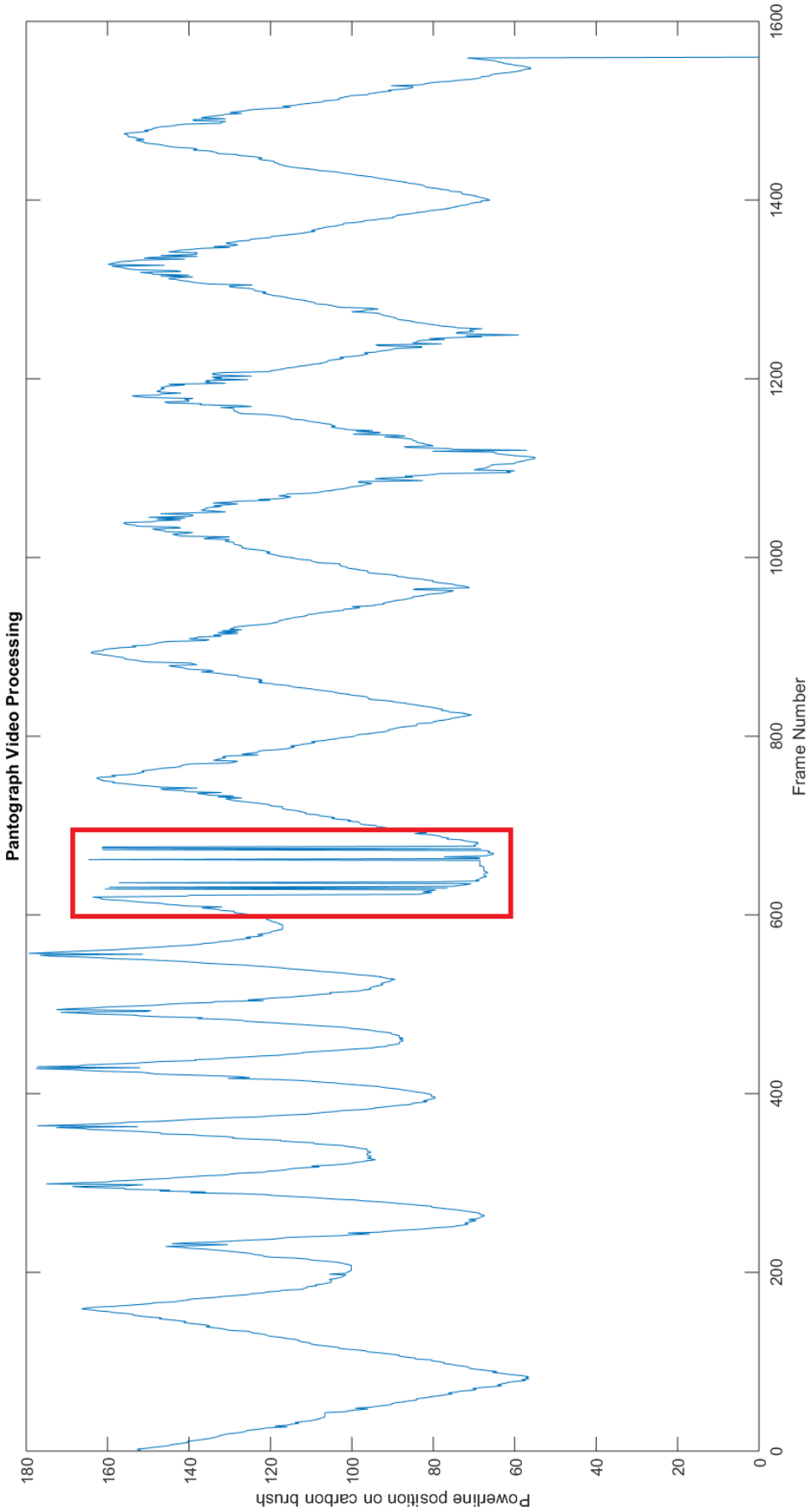

Figure 16: Power Line Transition Period

3.4.2 Possible Improvements

There are several avenues for improvement for this approach, namely:

- **Dynamic Temporal Thresholding:** Currently, a constant distance is used to threshold objects' movements in consecutive frames. However, given all objects conform to Newtonian motion, their position, velocity and acceleration must be continuous in time. This means the temporal threshold distance could be adjusted according to their previous velocity: as objects speed up the temporal threshold distance grows and vice versa.
- **Explicit Power Line Switching:** The event of switching power lines could be explicitly programmed into the approach. When four distinct vertical lines are identified, the line furthest from the previously classified power line is selected as the next power line. Given this, temporal thresholding could then also be used for the power lines.

4 Appendix A: Code Listings

4.1 License Plate Segmentation

Listing 1: License Plate Segmentation Script

```
theFigure = figure();

for carNumber = 1:26
    %Load image
    originalImage = imread(sprintf('numberplates/car%d.jpg', carNumber));
    originalImage = imresize(originalImage, 2);

    %Preprocessing
    grayImage = rgb2gray(originalImage);

    %Thresholding
    binarizedImage = ~imbinarize(grayImage, 0.5);

    %Define components in image and their size
    connectedComponents = bwconncomp(binarizedImage);
    numPixels = cellfun(@numel,connectedComponents.PixelIdxList);

    %Remove largest component
    [largestNum, idx] = max(numPixels);
    binarizedImage(connectedComponents.PixelIdxList{idx}) = 0;

    %Clear speckles
    for index = 1:length(connectedComponents.PixelIdxList)
        if(numPixels(index) < 20)
            binarizedImage(connectedComponents.PixelIdxList{index}) = 0;
        end
    end

    %Clear border
    binarizedImage = imclearborder(binarizedImage);

    %Morphologically merge number plate letters into blob
    binarizedImage = imfill(binarizedImage, 'holes');
    binarizedImage = imclose(binarizedImage, strel('line', 40, 0));
    binarizedImage = imfill(binarizedImage, 'holes');
    binarizedImage = imopen(binarizedImage, strel('line', 3, 0));
    binarizedImage = imopen(binarizedImage, strel('line', 3, 90));

    %Remove objects with incorrect plate dimensions
    minPlateDimRatio = 3;
    maxPlateDimRatio = 8.3;

    stats = regionprops('table',binarizedImage, 'MajorAxisLength', ...
                                'MinorAxisLength', 'PixelIdxList');
    componentDimensions = stats.MajorAxisLength ./ stats.MinorAxisLength;
```

```
46
47      for index = 1:length(componentDimensions)
48          if componentDimensions(index) < minPlateDimRatio || ...
49             componentDimensions(index) > maxPlateDimRatio
50              binarizedImage(stats.PixelIdxList{index}) = 0;
51          end
52      end
53
54      %Redefine components in image and their size
55      connectedComponents = bwconncomp(binarizedImage);
56      numPixels = cellfun(@numel,connectedComponents.PixelIdxList);
57
58      [largestNum, idx] = max(numPixels);
59
60      %Remove all but largest component
61    for index = 1:length(connectedComponents.PixelIdxList)
62          if index ~= idx
63              binarizedImage(connectedComponents.PixelIdxList{index}) = 0;
64          end
65      end
66
67      %Outline Convex Hull of number plate
68      stats = regionprops('table',binarizedImage, 'ConvexHull');
69
70      outline = transpose(cell2mat(stats.ConvexHull));
71      outline = reshape(outline, 1, numel(outline));
72      labelledImage = insertShape(originalImage, 'line', outline, ...
73                                              'LineWidth', 3, 'Color', 'red');
74
75      imshow(labelledImage);
76
77      waitforbuttonpress;
78      cla(gca);
79  end
80
81  close all;
```

4.2 Pantograph Video Processing

Listing 2: Pantograph Video Processing Class

```
1  classdef PantographVideoProcessor < VideoProcessor
2      properties (Constant)
3          VERTICAL_RANGE = 2:485;
4          HORIZONTAL_RANGE = 9:711;
5          cropFrame = @(originalFrame) ...
6              originalFrame(PantographVideoProcessor.VERTICAL_RANGE, ...
7                            PantographVideoProcessor.HORIZONTAL_RANGE,:);
8          thresholdFrame = @(aFrame, threshold) ...
9              uint8(255 * ~imbinarize(rgb2gray(aFrame), threshold));
```

```
10
11          TEMPLATE_CHANGE_THRESHOLD = 20;
12          templateToLeftEdge = [-86 -327];
13          templateToRightEdge = [-86 -101];
14      end
15
16      properties
17          theTemplateImage;
18          theLastTemplatePos = [0 0];
19          theCarbonBrushes;
20          theLeftIntersections;
21          theRightIntersections;
22          theTouchingIntersections;
23      end
24
25      methods
26          function self = PantographVideoProcessor(aVideoReader,
                aVideoWriter)
27              self@VideoProcessor(aVideoReader, aVideoWriter);
28              self.theTemplateImage = imread('pantograph_template.png');
29
30              self.theCarbonBrushes = zeros(self.theNumFrames, 4);
31              self.theTouchingIntersections = zeros(self.theNumFrames, 1);
32          end
33
34          function processedFrame = processingFunction(self, aCurrentFrame,
                aNextFrame)
35              aCroppedFrame = PantographVideoProcessor.cropFrame(
                    aCurrentFrame);
36              aCroppedNextFrame = PantographVideoProcessor.cropFrame(
                    aNextFrame);
37              processedFrame = self.finalApproach(aCroppedFrame,
                    aCroppedNextFrame);
38          end
39
40          function markTheSpot = finalApproach(self, aCurrentFrame,
                aNextFrame)
41              %%%% Crossbar detection %%%%
42              [carbonBrush, templatePosition] = self.carbonBrushDetection(
                    aCurrentFrame);
43              self.theCarbonBrushes(self.theFrameNumber, :) = carbonBrush;
44
45              %%%%Mark the crossbar
46              markTheSpot = insertMarker(aCurrentFrame, templatePosition
                    ([2, 1]),'Size',3,'Color','red');
47              markTheSpot = insertMarker(markTheSpot, carbonBrush([2, 1]),'
                    Size',3,'Color','red');
48              markTheSpot = insertMarker(markTheSpot, carbonBrush([4, 3]),'
                    Size',3,'Color','red');
```

```
49              markTheSpot = insertShape(markTheSpot, 'line', carbonBrush
                    ([2, 1, 4, 3]), 'LineWidth',2,'Color','green');
50
51              %%%% Define power line search section as above carbon brush
52              verticalSearchSection = 1:carbonBrush(1);
53              horizontalSearchSection = (carbonBrush(2):carbonBrush(4));
54              searchSectionOffset = [horizontalSearchSection(1) 0];
55              aCurrentFrameSearchSection = aCurrentFrame(
                    verticalSearchSection, ...
56                                                      horizontalSearchSection
                                                        ,:);
57
58              %%%%Create skeleton of powerlines in search section
59              linedSearchSection = ~imbinarize(rgb2gray(
                    aCurrentFrameSearchSection), 0.3);
60              linedSearchSection = uint8(255 * bwskel(linedSearchSection));
61              linedSearchSection = imdilate(linedSearchSection, strel('
                    rectangle', [3 1]));
62
63              %%%%Find & plot power lines
64              [houghMatrix, theta, rho] = hough(linedSearchSection);
65
66              %%%%Select relevant angles
67              lineDeviation = 15;
68              houghMatrix = houghMatrix(:, 90 - lineDeviation:90 +
                    lineDeviation);
69              theta = theta(:, 90 - lineDeviation:90 + lineDeviation);
70
71              P  = houghpeaks(houghMatrix, 10, 'NHoodSize', [15 15]);
72              lines = houghlines(linedSearchSection, theta, rho, P, '
                    MinLength', 0.5* carbonBrush(1));
73
74              %Plot power line
75              markTheSpot = plotLines(markTheSpot, lines,
                    searchSectionOffset);
76
77              %Find & plot intersection points
78              if(length(lines) == 0)
79                  self.theTouchingIntersections(self.theFrameNumber) = ...
80                      self.theTouchingIntersections(self.theFrameNumber -
                            1);
81              elseif length(lines) == 1
82                  intersections = findIntersection(lines, carbonBrush,
                        searchSectionOffset);
83                  self.theTouchingIntersections(self.theFrameNumber) = ...
84                                              intersections(1) - carbonBrush(2)
                                                    ;
85              elseif length(lines) >= 2
86                  [value, index] = max(abs(P(:,2)));
87                  lines = lines(index);
```

```
88                  intersections = findIntersection(lines, carbonBrush,
                        searchSectionOffset);
89                  self.theTouchingIntersections(self.theFrameNumber) = ...
90                                          intersections(1) - carbonBrush(2)
                                                ;
91              end
92
93              markTheSpot = insertMarker(markTheSpot, [carbonBrush(2) + ...
94                          self.theTouchingIntersections(self.theFrameNumber
                                ) ...
95                                  carbonBrush(1)], 'Size',4,'Color','blue')
                                     ;
96          end
97
98          function [carbonBrush, templatePosition] = carbonBrushDetection(
                self, aFrame)
99              %Threshold image to outline pantograph
100             binarizedFrame = PantographVideoProcessor.thresholdFrame(
                    aFrame, 0.3);
101
102             %Morphological processing
103             horizontalStrel = strel('line', 6, 0);
104             verticalStrel = strel('line', 12, 90);
105             edgedImage = imclose(binarizedFrame, horizontalStrel);
106             dilatedEdgeFrame = imdilate(edgedImage, verticalStrel);
107             cleanedEdgeImage = imclose(dilatedEdgeFrame,
                    verticalStrel);
108             thinnedEdgeImage = imerode(cleanedEdgeImage,
                    verticalStrel);
109
110             %Template matching
111             correlation = xcorr2(thinnedEdgeImage, self.
                    theTemplateImage);
112             [row, col] = find(correlation == max(max(correlation)),
                    1);
113             templatePosition = [row(1) col(1)];
114
115             %Temporal movement thresholding
116             if (sum(self.theLastTemplatePos) == 0) || ...
117                     (pdist2(self.theLastTemplatePos, templatePosition
                            ) ...
118                         < self.TEMPLATE_CHANGE_THRESHOLD)
119                 self.theLastTemplatePos = templatePosition;
120             else
121                 templatePosition = self.theLastTemplatePos;
122             end
123
124             %Locate carbon brush relative to template
125             carbonBrush = [templatePosition + self.templateToLeftEdge
                    ...
```

```
                                templatePosition + self.
                                    templateToRightEdge];
            end
    end

    methods (Static)
        function bwImageOutline = morphologicalApproach(aFrame)
            %Threshold image to outline pantograph
            binarizedFrame = PantographVideoProcessor.thresholdFrame(
                aFrame, 0.3);

            %Morphological processing
            horizontalStrel = strel('line', 6, 0);
            verticalStrel = strel('line', 12, 90);
            edgedImage = imclose(binarizedFrame, horizontalStrel);
            dilatedEdgeFrame = imdilate(edgedImage, verticalStrel);
            cleanedEdgeImage = imclose(dilatedEdgeFrame, verticalStrel);
            bwImageOutline = imerode(cleanedEdgeImage, verticalStrel);
        end

        function bwImageOutline = edgeDetectionApproach(aFrame)
            binarizedFrame = PantographVideoProcessor.thresholdFrame(
                aFrame, 0.3);
            bwImageOutline = edge(binarizedFrame, 'canny');
        end

        function bwImageOutline = temporalDiffApproach(aCurrentFrame,
            aNextFrame)
            frameDifference = imbinarize(rgb2gray(aNextFrame) - ...
                                         rgb2gray(aCurrentFrame), 0.2)
                                            ;
            bwImageOutline = imdilate(frameDifference, strel('line', 3,
                0));
        end
    end
end
```

Listing 3: Video Processing Base Class

```
classdef VideoProcessor < handle
    properties
        theVideoReader;
        theVideoWriter;
        theCurrentFrame;
        theNextFrame;
        theFrameNumber = 1;
        theNumFrames;
    end

    methods
        function self = VideoProcessor(aVideoReader, aVideoWriter)
```

```
            self.theVideoReader = aVideoReader;
            self.theVideoWriter = aVideoWriter;
            self.theNumFrames = aVideoReader.Duration * aVideoReader.
               FrameRate;

            if self.hasNext()
                self.theCurrentFrame = self.theVideoReader.readFrame();
            end
        end

        function bool = hasNext(self)
            bool = self.theVideoReader.hasFrame();
        end

        function processFrame(self)
            self.theNextFrame = self.theVideoReader.readFrame();

            myProcessedFrame = self.processingFunction(self.
               theCurrentFrame, self.theNextFrame);

            self.theVideoWriter.writeVideo(myProcessedFrame);
            self.theCurrentFrame = self.theNextFrame;
            self.theFrameNumber = self.theFrameNumber + 1;
        end

        function processVideoToEnd(self)
            while self.hasNext()
                self.processFrame();
            end
        end
    end

    methods (Abstract)
        processedFrame = processingFunction(aCurrentFrame, aNextFrame);
    end
end
```

Listing 4: Video Processing Script

```
theVideoFilename = 'Eric_Video';
theVideoAnnotation = '_labelled';
theVideoFiletype = '.avi';

theVideoReader = VideoReader(strcat(theVideoFilename, theVideoFiletype));
theVideoWriter = VideoWriter(strcat(theVideoFilename, theVideoAnnotation,
    theVideoFiletype));

open(theVideoWriter);

theVideoProcessor = PantographVideoProcessor(theVideoReader,
   theVideoWriter);
```

```
theVideoProcessor.processVideoToEnd();

close(theVideoWriter);

plot(theVideoProcessor.theTouchingIntersections);
title("Pantograph Video Processing");
xlabel("Frame Number");
ylabel("Powerline position on carbon brush");
```

4.3 Other

Listing 5: Localised Thresholding

```
function [processedImage] = blurThresholding(originalImage, blurrerSize)

    if nargin < 2
        blurrerSize = 60;
    end

    blurrerKernel = ones(blurrerSize, blurrerSize) ./ (blurrerSize .^ 2);

    grayImage = rgb2gray(originalImage);
    thresholdAverages = conv2(grayImage, blurrerKernel, 'same');

    max(max(grayImage))
    max(max(thresholdAverages))

    processedImage = grayImage > thresholdAverages;

end
```

Listing 6: Intersection Calculation

```
function [intersects] = findIntersection(lines, carbonBrush, offset)
if nargin < 3
    offset = [0 0];
end

carbonBrushY = mean(carbonBrush([1, 3]));

intersects = zeros(length(lines), 2);
    for i = 1:length(lines)
        pointA = lines(i).point1;
        pointB = lines(i).point2;

        %Creates a x = my + c model for the line
        linePoly = polyfit([pointA(2) pointB(2)], [pointA(1) pointB(1)],
           1);

```

```
16          intersects(i,:) = [polyval(linePoly, carbonBrushY) carbonBrushY]
               + offset;
17      end
18  end
```