

# ASSIGNMENT 1

ELEC4630: IMAGE PROCESSING AND COMPUTER VISION

---

William E. G. Kvaale, University of Queensland

30/03/2020

## Automatic Number Plate Recognizer

The first of many challenges I first realized I was facing, was the difference in resolution and image clarity (spatial resolution). The distortion (or skewness) in relation to the number plates was also a factor I thought would be challenging to address. The variation in lighting conditions did not vary too much, however, the challenge of having different colored number plates got me thinking I would soon be wandering in to [HSV](#)- or [LAB](#) space.

### Different approaches

Before diving into attempting to solve the problem at hand, I did some research on what technologies and approaches other researchers had done. Mainly I sought out older research papers, since most of the fresher papers that I found on [Google Scholar](#) was using some Machine Learning (specifically Deep Learning) approaches.

The key takeaway I was took, was that a Python approach combined with the OpenCV library and some sort of morphological approach might be a way to obtain success.

My first approach to the problem of different resolution, was resizing the images. This was done in the pursuit of transforming the data to a somewhat normalized format. I read several papers advising against, and some stating this could help achieve better result in regards to edge detection tasks. I went found that a suitable minimum width and height was respectively 720 pixels and 1280 pixels. See Listing 1.

The interpolation used in resizing, was [OpenCV's mystic cv.INTER\\_AREA](#). OpenCV states that it is "resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER\_NEAREST method". In layman's term:

1. **Scale down:** is either a bilinear interpolation with (1,0) as coefficients, or more complicated if both the input and output scales, in regards to width and height, are not integers
2. **Scale up:** is equal to linear interpolation and exact linear interpolation based on the number of channels

I also experimented with the lightness of the images, or more concretely I tried a decoding gamma ( $\gamma > 1$ ) on the images, but empirically I found that an encoding gamma (however so

slightly) ( $\gamma < 1$ ) improved performance. I set  $\gamma = 0.9$  throughout my different approaches.

---

```
1 def checkSize(image):
2     height, width = image.shape[:2]
3     if height < MIN_HEIGHT:
4         return imutils.resize(image, height=MIN_HEIGHT,
5                               inter=cv.INTER_AREA)
6     elif width < MIN_WIDTH:
7         return imutils.resize(image, width=MIN_WIDTH,
8                               inter=cv.INTER_AREA)
9     else:
10        return image
```

---

Listing 1: checkSize function in Python

To segment the image, I tested several threshold algorithms; binary threshold, tozero threshold, triangle threshold, adaptive thresholding and Otsu's Binarization. The method proving best in terms of segmenting the images the most sufficient was Otsu's. Otsu's finds the optimal threshold value, instead of having it set manually with plain binary thresholding.

On beforehand of applying the threshold, I would smooth the images using one of OpenCV's built in filters;

`cv.blur()`, `cv.GaussianBlur()`, `cv.bilateralFilter()`

The one I had the most success with was the Gaussian Blur.

## **Method №1**

1. Resize image
2. Encode image with  $\gamma = 0.9$
3. Smooth using Gaussian with 5x5 kernel
4. Otsu's threshold to segment image
5. Bitwise AND threshold with original image
6. Run Canny Edge on the preceding output

My initial approach, which achieved 4 out of 8.



Figure 1: Results using №1

## **Method №2**

1. Resize image
2. Encode image with  $\gamma = 0.9$
3. Smooth using Gaussian with 5x5 kernel
4. Adaptive Gaussian Threshold
  - 4.1. Block size = 15
  - 4.2. C = -3
5. Morphological operations
  - 5.1. kernel = 2x2 rectangular structuring element
  - 5.2. 1 iteration with closing
  - 5.3. 1 iteration using opening

My second approach, which achieved 4 out of 8. However, this time it failed on the Ferrari - but managed to locate the numberplate on the Audi A4.

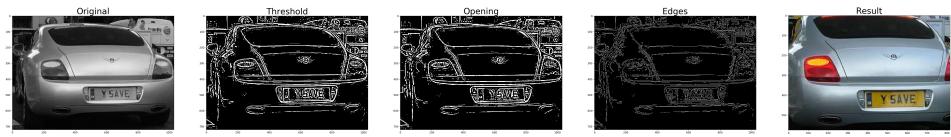
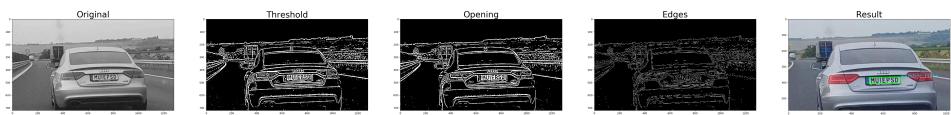


Figure 2: Results using method №2

### **Method №3**

1. Resize image
2. Encode image with  $\gamma = 0.9$
3. Smooth using Gaussian
  - 3.1. 7x7 kernel
  - 3.2.  $\sigma_x = 2$
4. Adaptive Gaussian Threshold
  - 4.1. Block size = 15
  - 4.2. C = -3
5. Bitwise AND threshold with original image

My third and final approach, which achieved 5 out of 8. This also fails on the Ferrari, and the Bentley and Police car.

I never managed to detect the police car accurately, nor the Bentley. The Police car was quite difficult, but could maybe be achieved through template matching (using an exact template). The Bentley I could detect using a different contouring method than all the others, which I found interesting. It was successful by using [OpenCV's cv.boundingRect](#).

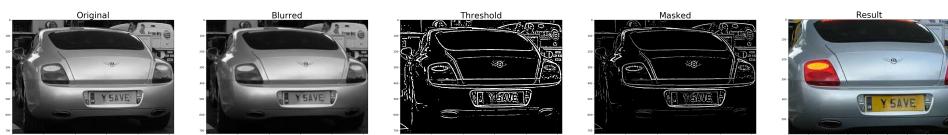
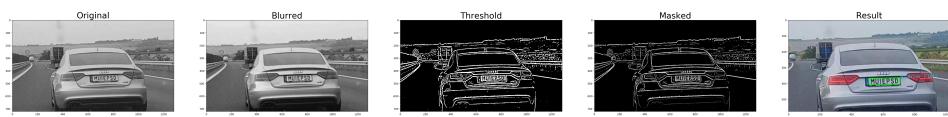


Figure 3: Results using method №3

## Pantograph Tracking and Position Graphing

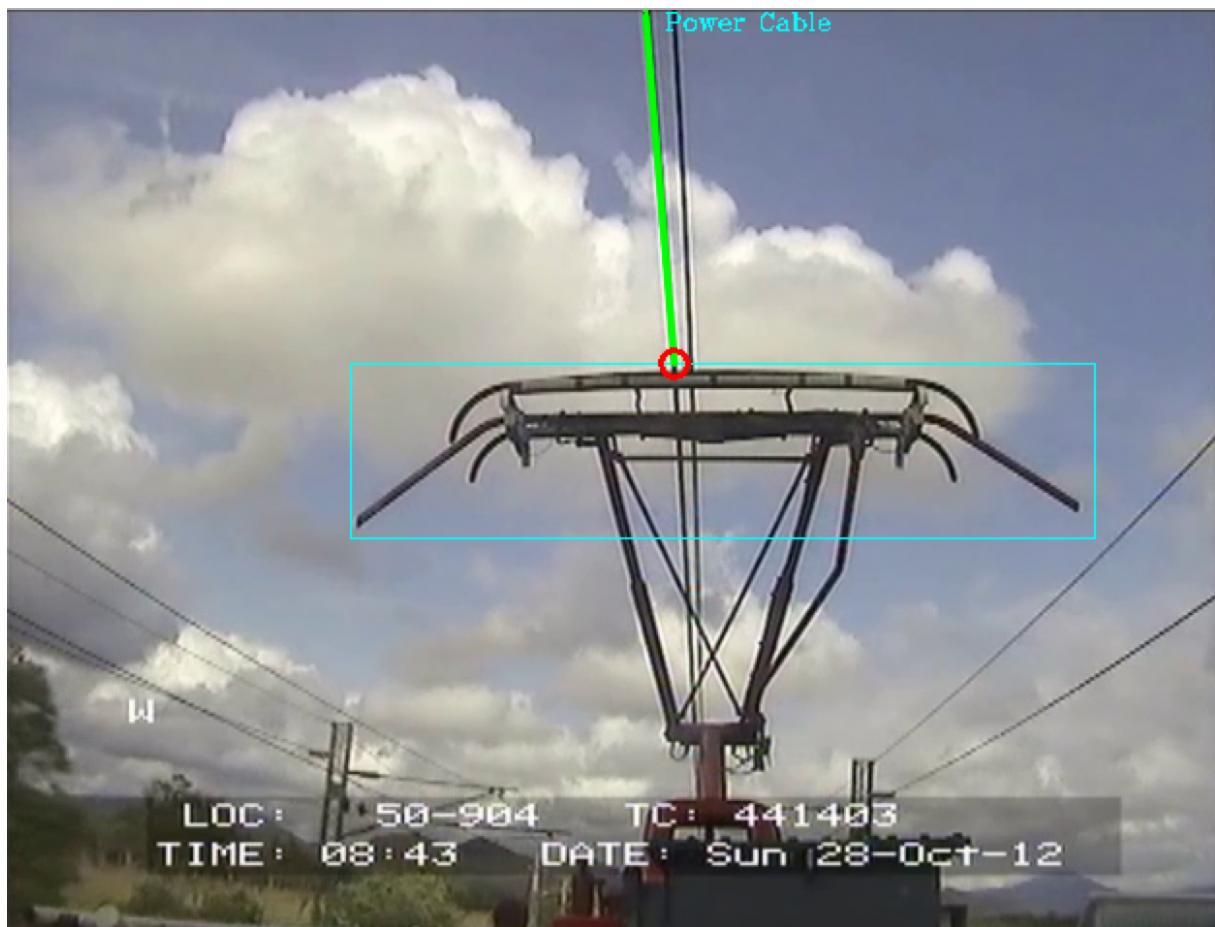


Figure 4: Pantograph tracking using MOSSE

This was my first time ever working with videofiles, and tracking - and I was quite excited upon starting working on this part of the assignment. I quickly realized that working on videofiles is not that different from working with still images, if you handle it frame by frame.

OpenCV delivers several object trackers; BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE and CSRT. For my case, I found MOSSE being the best fit. MOSSE (Minimum Output Sum of Squared Error) utilizes adaptive correlation and outputs stable correlation filters (as long as it is initialized with one frame - preferably the ROI).

In regards to the differentiate the suspension cable and power cable, I had a rather simple approach. By observation, it would appear thicker, that being having a larger circumference, or being closer to the camera, was not at all time clear. Regardless, I used this as my selection criteria.

## Different approaches

By following the lectures, my initial thought was that success could most certainly be achieved by using a standard or probabilistic Hough transformation to find the lines, and the intersections of such lines. Nevertheless, my attempts in implementing this did not bear fruit, and therefore I chose to go down a different path - a morphological approach. I got inspired by [OpenCV's tutorial on line detection](#), and used this as a baseline for my approach to achieve the best results.

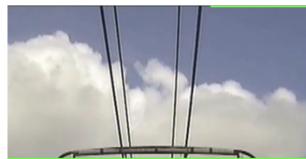


Figure 5: Hough transformation (standard)

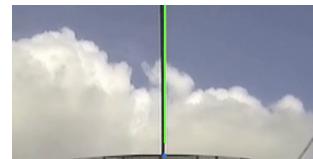


Figure 6: Hough transformation (probabilistic)

Even though the probabilistic Hough transformation could yield a good fit, it was highly unstable to noise, such as lines crossing.

## Successful method

1. Convert to grayscale
2. Adaptive Gaussian Threshold
  - 2.1. Block size = 15
  - 2.2. C = -3
3. Morphological operations
  - (a) Vertical structuring element of dimensions 1x16
  - (b) Erode 1 iteration
  - (c) Dilate 2 iterations
4. Smooth image using bilateral filtering
  - (a) diameter = 11
  - (b)  $\sigma_{color} = 17$
  - (c)  $\sigma_{space} = 17$
5. Run Canny Edge on the preceding output

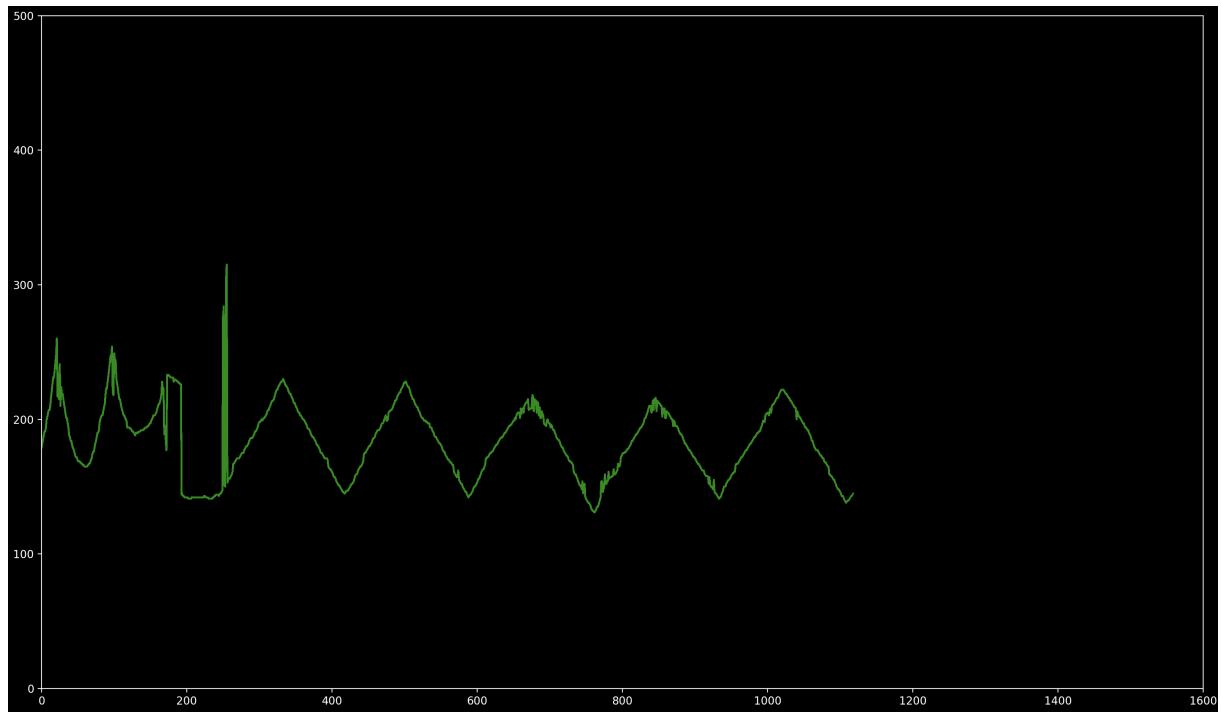


Figure 7: Graphing the position of intersection

### Challenges to pantograph detection

One of the main challenges was to get rid of the clouds in the background, and be left with a clean good shot of the contours of the cables. If the sky was always cloud free, this would not have been an issue - but this is not La La Land. During the night I would assume one would either have the cable painted with some self illuminating matter, or have improve lighting conditions using spotlights. Heavy could also be of a problem, since it might be recorded on the camera as a *line*. Varying the structuring element might be an approach here to avoid detecting *lines of rain*.



Figure 8: Detecting left power cable



Figure 9: Detecting right power cable

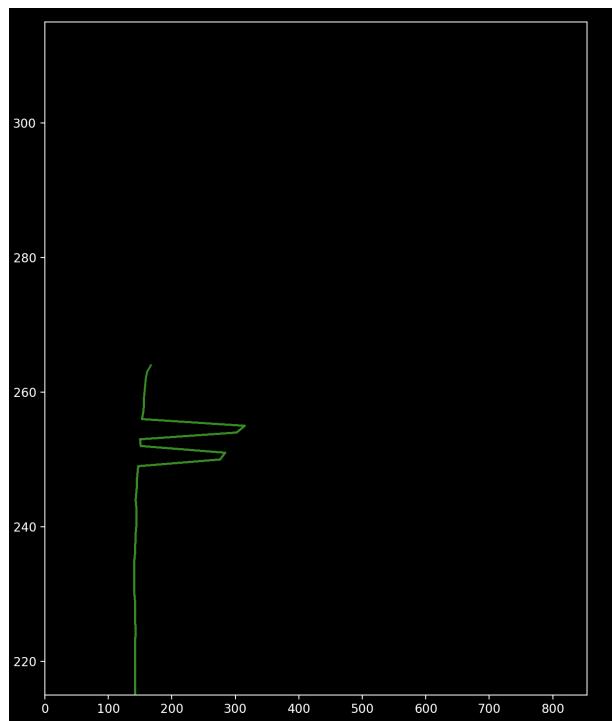


Figure 10: Flickering due to two simultaneous power cables

As a result of having two visible power cables simultaneously, there was a flicker in the graphing. This could be handled by creating a rule based system, however, I do not think it is easy to generalize this situation and I do not have the domain knowledge regarding which it should *choose*.



Figure 11: The ROI for tracking power cables

This is the view of the processed ROI. As one can see, it is not perfect, but it works very well in my opinion. I was unable to denoise any further without affecting the power- and suspension cable, which is undesirable. I could might improve my solution by applying some morphological operations to remove unwanted lines (noise).

## Sources

[https://docs.opencv.org/master/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html)

[https://en.wikipedia.org/wiki/Gamma\\_correction](https://en.wikipedia.org/wiki/Gamma_correction)

<https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>

<https://docs.opencv.org/>

<https://scikit-image.org/docs/dev/>

The entire sourcecode for this project can be found in [my GitHub repository](#).

## Code

---

```
1 import os
2 import glob
3 import imutils
4 import cv2 as cv
5 import numpy as np
6 from skimage import feature
7 from skimage import img_as_ubyte
8 from matplotlib import pyplot as plt
9 from datetime import datetime
10
11 FILEPATH = os.path.abspath("images")
12 OUTPUT = os.path.abspath(("output"))
13
14 MIN_HEIGHT = 720
15 MIN_WIDTH = 1280
16 MIN_THRESH = 192
17 MAX_THRESH = 255
18
19
20 def saveImages(destination, images, label=""):
21     for i in range(len(images)):
22         img = images[i]
23         try:
24             cv.imwrite(f"{destination}/out_car_{i}_{label}.jpg",
25             cv.cvtColor(img, cv.COLOR_RGB2BGR))
26             print(f"Saving number plate {i+1} \U0001F4BE")
27         except IOError :
28             return f"Error while saving file number: {i}"
29     print(f"Successfully saved {len(images)} images to {destination}")
30
31 def loadImages(filepath, carNumber=' '):
32     images = []
33     for file in sorted(glob.glob(f"{filepath}/*.jpg")):
34         if file.lower().endswith('.jpg'):
35             img = cv.imread(file, 1)
36             img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
37             images.append(img)
38     if carNumber != ' ' and isinstance(carNumber, int):
39         try:
40             return images[carNumber]
41         except:
```

```

42         return f"Can't find that image of carNumber: {carNumber}"
43     return images
44
45
46 def checkSize(image):
47     (height, width) = image.shape[:2]
48     if height < MIN_HEIGHT:
49         return imutils.resize(image, height=MIN_HEIGHT)
50     elif width < MIN_WIDTH:
51         return imutils.resize(image, width=MIN_WIDTH)
52     else:
53         return image
54
55
56 def getContourpApproximate(image, edges):
57     contours, _ = cv.findContours(edges.copy(), cv.RETR_TREE,
58                                   cv.CHAIN_APPROX_SIMPLE)
59     updated_contours = sorted(contours, key=cv.contourArea, reverse=True)[:10]
60
61     for cnt in updated_contours:
62         perimeter = cv.arcLength(cnt, True)
63         approx = cv.approxPolyDP(cnt, 0.04 * perimeter, True)
64         x = approx.ravel()[0]
65         y = approx.ravel()[1]
66         padding = 1
67         if len(approx) == 4:
68             cv.putText(image, "Number Plate", (x + padding, y + padding),
69                         cv.FONT_HERSHEY_COMPLEX, 1, (0, 255, 255))
70             cv.drawContours(image, [approx], -1, (0, 255, 0), 3,
71                           lineType=cv.LINE_AA)
72             print(f"Added a candidate! \U0001F919 ")
73             break
74
75
76 def getContoursBoundingRectangle(image, edges):
77     contours, _ = cv.findContours(edges.copy(), cv.RETR_EXTERNAL,
78                                   cv.CHAIN_APPROX_SIMPLE)
79     updated_contours = sorted(contours, key=cv.contourArea, reverse=True)[:3]
80
81     cv.drawContours(image, contours, -1, (255, 0, 0), 2)
82     for cnt in updated_contours:
83         x, y, w, h = cv.boundingRect(cnt)
84
85         ratio = float(h)/w
86         if ratio < 0.5 and ratio > 0.18:

```

```

83
84         cv.rectangle(image,(x,y),(x+w,y+h),(0,255,0), 3)
85
86
87     def getCrop(image, factor=1):
88         img = image.copy()
89         h, w = img.shape[:2]
90         return img[factor*(h//10):(10-factor)*(h//10),
91                    factor*(w//10):(10-factor)*(w//10)]
92
93
94     def getEdges(image, opencv=True):
95         img = image.copy()
96         if opencv:
97             return cv.Canny(img, 70, 200)
98         return feature.canny(img, sigma=2)
99
100
101
102     def adjustGamma(image, gamma=1.0):
103         # source: https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/
104         img = image.copy()
105         inverted_gamma = 1.0 / gamma
106         look_up_table = np.array([(i / 255.0) ** inverted_gamma * 255
107                               for i in np.arange(0, 256)]).astype("uint8")
108         return cv.LUT(img , look_up_table)
109
110
111     def saveFigures(images, titles=[], rows=1):
112         now = datetime.now()
113         columns = len(images)
114         _ = plt.figure(figsize=(64, 64/columns))
115         for i in range(1, columns * rows + 1):
116             plt.subplot(1, columns, i)
117             if not titles[i-1]:
118                 plt.gca().set_title(f"Subplot_{i-1}", fontsize=32)
119             plt.gca().set_title(titles[i-1], fontsize=32)
120             plt.imshow(images[i-1], cmap='gray')
121             plt.savefig(f"figures/figure_{i}-{now}.png")
122             print(f"Saved a figure \U0001F4BE")
123
124     def showFigures(images, titles=[], rows=1):
125         columns = len(images)
126         fig = plt.figure(figsize=(32, 32/columns))
127         for i in range(1, columns*rows+1):
128             fig.add_subplot(rows, columns, i)

```

```

127     if not titles[i-1]:
128         plt.gca().set_title(f"Subplot_{i-1}")
129     plt.gca().set_title(titles[i-1])
130     plt.imshow(images[i-1], cmap='gray')
131     plt.show()
132
133
134 def attemptOne(image):
135     orig_img = checkSize(image)
136     img = cv.cvtColor(orig_img.copy(), cv.COLOR_RGB2GRAY)
137
138     gammad = adjustGamma(img, gamma=0.9)
139     blurred = cv.GaussianBlur(gammad, (5,5), 0)
140     _, threshold = cv.threshold(blurred, MIN_THRESH, MAX_THRESH, cv.THRESH_OTSU)
141
142     mask = cv.bitwise_and(img, threshold)
143
144     edges = img_as_ubyte(geCannyEdges(mask))
145
146     getContourpApproximate(orig_img, edges)
147     return orig_img
148
149
150 def attemptTwo(image):
151     orig_img = checkSize(image)
152     img = cv.cvtColor(orig_img.copy(), cv.COLOR_RGB2GRAY)
153     adjusted = adjustGamma(img, gamma=0.9)
154     blurred = cv.GaussianBlur(adjusted, (5, 5), 0)
155
156     _, threshold = cv.threshold(blurred, MIN_THRESH, MAX_THRESH,
157                                 cv.THRESH_BINARY + cv.THRESH_OTSU)
158     mask = cv.bitwise_and(threshold, threshold)
159
160     kernel = cv.getStructuringElement(cv.MORPH_RECT, (3,3))
161     erode = cv.erode(mask, kernel)
162     closing = cv.morphologyEx(erode, cv.MORPH_CLOSE, kernel)
163     opening = cv.morphologyEx(closing, cv.MORPH_OPEN, kernel)
164
165     getContoursBoundingRectangle(orig_img, opening)
166     return orig_img
167
168 def attemptThree(image):
169     orig_img = checkSize(image)
170     img = cv.cvtColor(orig_img.copy(), cv.COLOR_RGB2GRAY)

```

```

171     adjusted = adjustGamma(img, gamma=0.9)
172     blurred = cv.medianBlur(adjusted, 5)
173
174     threshold = cv.adaptiveThreshold(blurred ,255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
175                                     cv.THRESH_BINARY,11,2)
176     mask = cv.bitwise_and(img, threshold)
177     denoise = cv.fastNlMeansDenoising(mask, h=10, templateWindowSize=7,
178                                     searchWindowSize=21)
179
180     edges = img_as_ubyte(getCannyEdges(denoise, opencv=False))
181
182
183
184     def attemptFour(image):
185         orig_img = checkSize(image)
186         orig_img = getCrop(orig_img)
187         img = cv.cvtColor(orig_img.copy(), cv.COLOR_RGB2GRAY)
188
189         gammad = adjustGamma(img, gamma=0.9)
190         blurred = cv.GaussianBlur(gammad, (7,7), -3)
191         #_, threshold = cv.threshold(blurred, MIN_THRESH, MAX_THRESH,
192         #                           cv.THRESH_OTSU)
193         threshold = cv.adaptiveThreshold(blurred, 255,
194                                         cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 15, -3)
195
196         mask = cv.bitwise_and(img, threshold)
197
198         edges = img_as_ubyte(getCannyEdges(mask))
199         #save_figures([img, threshold, mask, edges], titles=["Original",
200         #           "Thresholded", "Masked", "Canny Edged"])
201         #show_figures([img, threshold, mask, edges], titles=["Original",
202         #           "Threshold", "Masked", "Edged"])
203
204     def main():
205         images = loadImages(FILEPATH)
206         out_images = []
207
208         for image in images:
209             output = attemptOne(image)

```

```
210         out_images.append(output)
211
212     saveImages(OUTPUT, out_images, label="290320")
213
214 main()
```

---

```
1 import cv2 as cv
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 def getCrop(frame, point1, point2):
7     img = frame.copy()
8     padding = 1 # 1 pixel for padding
9     return img[0:point1[1] + padding, point1[0]:point2[0] - padding]
10
11
12 def houghlineProbabilisticProcessFrame(frame):
13     gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
14     gray_copy = np.copy(gray)
15     edges = cv.Canny(gray_copy, 50, 150, apertureSize=3)
16     lines = cv.HoughLinesP(edges, 1, np.pi/180, 100, 100, 100)
17     try:
18         for x1, y1, x2, y2 in lines[0]:
19             cv.line(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
20     except:
21         pass
22
23
24 def houghlineStandardProcessFrame(frame):
25     gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
26     lines = np.copy(gray)
27     edges = cv.Canny(lines, 70, 200, apertureSize=3)
28     lines = cv.HoughLines(edges, 1, np.pi/180, 192, None, 0)
29
30     if lines is not None:
31         for i in range(0, len(lines)):
32             rho = lines[i][0][0]
33             theta = lines[i][0][1]
34             a = np.cos(theta)
35             b = np.sin(theta)
```

```

36         x0 = a * rho
37         y0 = b * rho
38         pt1 = (int(x0 + 1000 * (-b)), int(y0 + 1000 * (a)))
39         pt2 = (int(x0 - 1000 * (-b)), int(y0 - 1000 * (a)))
40         cv.line(lines, pt1, pt2, (0, 0, 255), 3, cv.LINE_AA)
41     return lines
42
43
44 def morphProcessFrame(frame):
45     # source:
46     #   https://docs.opencv.org/3.4/dd/dd7/tutorial_morph_lines_detection.html
47     verticalSize = 16
48     verticalStructure = cv.getStructuringElement(cv.MORPH_RECT, (1,
49     ↪   verticalSize))
50
51     gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
52
53     lines = np.copy(gray)
54     lines = cv.bitwise_not(lines)
55     lines = cv.adaptiveThreshold(lines, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
56     ↪   cv.THRESH_BINARY, 15, -3)
57     lines = cv.erode(lines, verticalStructure, iterations=1)
58     lines = cv.dilate(lines, verticalStructure, iterations=2)
59
60     smooth = np.copy(lines)
61     smooth = cv.bilateralFilter(smooth, 11, 17, 17)
62     rows, columns = np.where(smooth != 0)
63     lines[rows, columns] = smooth[rows, columns]
64
65     return lines
66
67
68 def getContour(orig_frame, edges, dx, dy):
69     contours, _ = cv.findContours(edges, cv.RETR_EXTERNAL,
70     ↪   cv.CHAIN_APPROX_SIMPLE)
71     power_cable = sorted(contours, key=cv.contourArea, reverse=True)[0]
72     epsilon = cv.arcLength(power_cable, True)
73     approx = cv.approxPolyDP(power_cable, epsilon*0.069, True)
74     x, y = approx.ravel()[0], approx.ravel()[1]
75     fontScale = (orig_frame.shape[0] * orig_frame.shape[1])/(np.power(10, 6))
76
77     cv.putText(orig_frame, "Power Cable", (x + dx + 10, y + dy + 10),
78     ↪   cv.FONT_HERSHEY_COMPLEX, fontScale, (255, 255, 0))
79     cv.drawContours(orig_frame, [approx], -1, (0, 255, 0), thickness=2,
80     ↪   lineType=cv.LINE_AA, offset=(dx, dy))

```

```

75     return approx
76
77
78 def setCrosshair(orig_img, contour, dx, dy):
79     x,y = contour[-1].ravel()[0], contour[-1].ravel()[1]
80     cv.circle(orig_img, (x + dx, y + dy), 7, (0, 0, 255), 2)
81
82
83 def getPosition(power_cable):
84     return power_cable[-1].ravel()[0]
85
86
87 def initGraph(style='dark_background'):
88     y_pos = 0
89     xs, ys = [], []
90     plt.style.use(style)
91     fig = plt.figure(figsize=(8, 10))
92     ax = fig.add_subplot(111)
93     ax.set_xlim(left=0, right=854) # width of videofile
94     fig.show()
95
96     return xs, ys, fig, ax, y_pos
97
98
99 def main():
100     # Initialize reading video file
101     videoCap = cv.VideoCapture("videos/Eric2020.mp4")
102     init_frame_read_correctly, init_frame = videoCap.read()
103     if not init_frame_read_correctly:
104         print("No frames to read...")
105
106     # Set Bounding Box for Region of Interest
107     bounding_box = (293, 173, 376, 87) # alternatively: use
108     ↳ cv.selectROI('tracking', init_frame)
109
110     # Set tracker and initialize "Toggle switch" for tracking
111     tracker = cv.TrackerMOSSE_create()
112     initial_kickstart = True
113
114     # Initialize graph, for position of intersection between pantograph and
115     ↳ power cable
116     xs, ys, fig, ax, y_pos = initGraph()
117
118     while videoCap.isOpened():
119         # Start reading video file after ROI is set

```

```

118     frame_read_correctly, frame = videoCap.read()
119     if not frame_read_correctly:
120         print("No frames to read...")
121         break
122
123     # Initialize the tracker
124     if initial_kickstart:
125         tracker.init(frame, bounding_box)
126         initial_kickstart = False
127
128     # Update bounding box for rectangle
129     frame_read_correctly, new_bounding_box = tracker.update(frame)
130     #print(f"x1: {new_bounding_box[0]}\t y1: {new_bounding_box[1]}\t x2:
131     ↳ {new_bounding_box[2]}\t y2: {new_bounding_box[3]}")
132     if frame_read_correctly:
133         point1 = (int(new_bounding_box[0]), int(new_bounding_box[1]))
134         point2 = (int(new_bounding_box[0] + new_bounding_box[2]),
135         ↳ int(new_bounding_box[1] + new_bounding_box[3]))
136         cv.rectangle(frame, point1, point2, (255, 255, 0), 1)
137
138     # Crop out new ROI and process it
139     new_ROI = getCrop(frame, point1, point2)
140     processed_crop = morphProcessFrame(new_ROI)
141
142     # Draw contour on the power cable, and set crosshair (red dot) at
143     ↳ intersection with pantograph
144     power_cable = getContour(frame, processed_crop, point1[0], 0)
145     setCrosshair(frame, power_cable, point1[0], 0)
146
147     # Update Graph
148     fig.canvas.draw()
149     ax.set_ylim(bottom=max(0, y_pos - 50), top=y_pos + 50)
150     ax.plot(xs, ys, color='g')
151     xs.append(getPosition(power_cable))
152     ys.append(y_pos)
153     y_pos += 1
154
155     # Show processed- and video frame
156     cv.imshow("Cropped", processed_crop)
157     cv.imshow("Pantograph", frame)
158
159     # Press 'q' to quit the windows
160     if cv.waitKey(1) & 0xFF == ord('q'):
161         break

```

```
160     # Release memory when job is finished  
161     videoCap.release()  
162     cv.destroyAllWindows()  
163  
164  
165 main()
```

---