INTRODUCTION TO COMPUTER VISION

VEHICLE LICENSE PLATE BLOCKER

December 8, 2019

Casey Duncan

Colorado School of Mines

INTRODUCTION

The license plate (LP) for a vehicle is like the face for a person, since it is typically the main attribute that distinguishes that specific vehicle from all other vehicles. This information is quite useful since it can be used to identify who the vehicle belongs to, if the vehicle has been registered, vehicle history, and several other attributes. Because of this, LPs are blurred in public images/videos on TV and in photos shown in Google Earth's street view. Additionally, a considerable amount of research has gone into Automatic License Plate Detection (ALPD) for traffic & toll violations and vehicle monitoring.

PREVIOUS WORK

History

ALPD was first developed in as early as 1976 within Britain for its Police Departments. However, these systems were quite expensive which hindered them from becoming widely using. In the 1990s, innovations in software and computation allowed for these systems to become more prevalent in law enforcement departments around the world. During this time, ALPR was implemented into mobile vehicles and became as simple to use as "point and shoot", capturing vehicle LPs at speeds greater than 100 mph. Common techniques used to tackle this problem, along with their success, will be discussed in the next section. Despite their success, the techniques used to detect the LPs still have difficulties, such as scenarios with poor lighting, low contrast due to shadows, a large variety of LP shapes & colors, and many more. Contrary to popular belief, ALPR is still a challenging problem due to these conditions and the research to achieve this task is continuing to be done.

Research

Boundary/Edge Detection

The most common method used to detect LPs is implementing methods to detect the edges of the LP. This is often done using a Sobel edge detector [2][5][6][7][9], which is a common method used to detect vertical edges in an image. This method is often used because vehicles usually have more horizontal lines than vertical lines, making the LP easier to detect. Next, the vertical edges are processing to find two parallel vertical edges, which form a rectangle. This can either be done by setting a parallelism threshold of using Hough Transforms [2][5][6]. Next, the rectangle's aspect ratio (width-to-height ratio) is checked. If this satisfies the aspect ratio of the common LP, a LP is likely detected. This approach is quite robust depending on the consistency in lighting conditions, and is great because it is simple, fast, and straightforward. However, it is hardly applied to complex images since unwanted edges are often detected. As you will see below, this is the initial approach I took.

Color Features

Another common method for detecting LPs is to search for specific color edges that only appear on the LP, since the color combination between the LP and the License Plate Characters (LPCs) is unique. One method used was searching for pixels that fall within a specific hue, saturation, & intensity while also forming a rectangle [2][3][4]. Another was searching for pixels that fall within an upper & lower threshold for LP brightness depending on images average brightness while also forming a rectangle [5]. Using color features as a detection method is great since it is not as reliant on the shape of the LP, but it can be quite computationally complex.

Character Features

LPs are also unique in that they all have a specific number of characters on them that are all the same font, meaning they have the roughly have the same shape. Therefore, if an algorithm can detect the LPCs, then it has found the LP. One method used to do this is to scan the image horizontally for repeating contrast changes that one would only expect to occur between the LP and the LPCs [2]. Another method used is to find objects that fit within a bounding box with a similar aspect ratio, since all LPCs will typically be the same size, and then apply a Hough Transform to the top & bottom of the objects to detect straight lines[5]. If these lines are parallel within a certain range, a LP is likely detected. As you will see below, this method is the most common to the method I implemented for this project. While detecting character features is robust to images viewing a LP at different angles, this method is time consuming and can result in errors if other text is present in image.

Neural Networks

As of 2017, deep Convolutional Neural Networks have been trained using over 20,000 images of vehicles with labeled LPs. The process and robustness of this is outlined in [8] and providing a successful detection rate of 100% on 348 images of US LPs.

Other Methods

Another common method used to detect LPs is using Connected Component Analysis (CCA) [1][10], which labels pixel colonies of 8 or more and then checks each colony for its spatial measurements (area, orientation, aspect ratio, etc.). In [1], the photo is inverted and checked again if no LP is detected, which has a success rate of 96.5% over 1334 images. One common problem with this method is that uneven illumination due to shadows or different weather conditions can break the LP up into multiple colonies of pixels. Therefore, [10] uses an improved Bernsen algorithm to create uniform lighting around LP.

LICENSE PLATE BLOCKER COMPONENTS

The technique required for blocking or blurring out a vehicle's LP requires two components, which are defined below.

1. License Plate Detector - Detects the location of the physical LP within the image or video frame of the vehicle. Reference the top image of Figure 1.

2. License Plate Masker - Apply a blur filter to the detected LP or overlay a masking image on top of the detected license plate. Reference the middle image of Figure 1.

For fun and because I had the required information, I also implemented a third component:

3. Vehicle-to-Camera Pose Estimator - Estimated the position (x,y,z) of the vehicle with respect to the camera based on the detected LP and the camera's intrinsic parameters. Reference the bottom image of Figure 1.

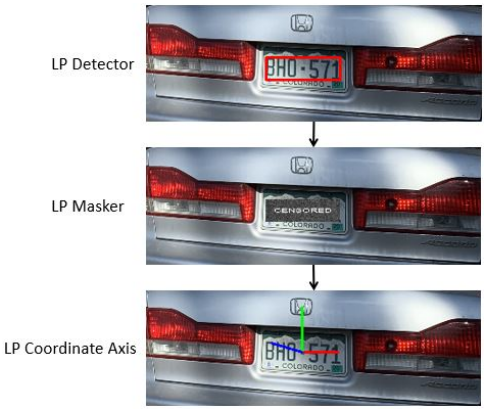

Figure 1: License Plate Blocker Components

I intended on using a Raspberry Pi 3 to implement this system on my personal vehicle, so I decided to use Python (version 3.7.4) and OpenCV (version 4.1.1) to build each component. In the below subsections, I will explain how I built each of these components with supplementary figures & pseudocode.

License Plate Detector

This was by far the most difficult component to build and is the one that has been researched the most. The reason this is so difficult to achieve is that LPs come in a large variety of shapes, colors, and designs, they sometimes can be obstructed by dirt or snow, and their visibility is affected by the environment that they are photographed in (night vs. day, low contrast due to shadows, etc.). Because of this, I tried various techniques to find the LP, many of which failed, and ended up going with the technique that found the LPCs rather than the actual LP. Additionally, it should be noted that I designed my algorithm using Colorado LPs as test images since this is what was most available to me. To visually illustrate the approaches I took, I will be using the image of my Honda Accord shown in Figure 2. It should be noted that all photos used to build the LP blocker were of size 1024×768 pixels.

Figure 2: Original Image of Vehicle

Unsuccessful Approaches

My original plan was to find the corners of the LP through the following steps:

1. Convert photo to black and white.
2. Detect edges by applying a variety of thresholding techniques to the photo, such as:
    - Changing all pixel values above or below a threshold value to be a value of 0 or 1.
    - Morphological Opening and/or Closing to accentuate edges
3. Find contours in image and filter through contours to find LP.
    - Because all LPs are rectangular, filter through out all contours that are not rectangular.
    - Because all LPs have characters within them, filter out all rectangular contours that do not include a certain number of contours within them.
4. Find the corners & center of the LP based on the LP contour's centroid & bounding box.

While this approach worked on some photos of LPs, it was not robust enough to handle most vehicles. This is because the contours associated with the LP would blend with other contours associated with the vehicle, such as the trunk on the rear or the radiator vent on the front. Also, there are many more contours than just the license plate, which can be seen in Figure 4. I was not expecting this, and this caused a lot of problems. To remove unwanted contours or separate contours, I tried tuning the following parameters:

- The threshold value.
- The blur parameters to remove the contours found in the trees.
- The edge detector parameters to remove the contours found in the trees and unwanted contours found on the car.
- Applying a variety of morphological transformations (closing, dilation, & gradient) on the edges photo to enhance the contours found on the license plate.

However, none of these methods helped the LP standout without making all the other contours standout more or destroyed the contours of the LP. One reason for this is that finding a threshold value that worked for multiple photos was impractical because the lighting & scenery changes in each photo. Additionally, the LP contours were not as strong as other contours found on the vehicle.

The next approach I took was using SURF to find similar features between a standard Colorado LP and the Colorado LP on a vehicle. Unfortunately, I did not have any success with this, and I believe it is because the characters on the standard Colorado were different from the LPCs on the vehicle.

Successful Approach

Because I had no success with detecting the actual LP, I decided to take the approach of detecting the LPCs. These were much easier to find since their contours were not mixing with other contours on the vehicle and were unique features compared to other features on the vehicle. To detect the bounding box enclosing the LPCs, I took the approach outlined below.

1. Convert photo to black and white.
2. Detect edges in image by applying a variety of thresholding techniques to the photo, such as:
    - Apply a filter to the photo to blur objects in the background, such as trees, buildings, or street signs. This was done using OpenCV's bilateralFilter() function.
    - Use OpenCV's Canny() function to detect edges in image. Reference Figure 3.

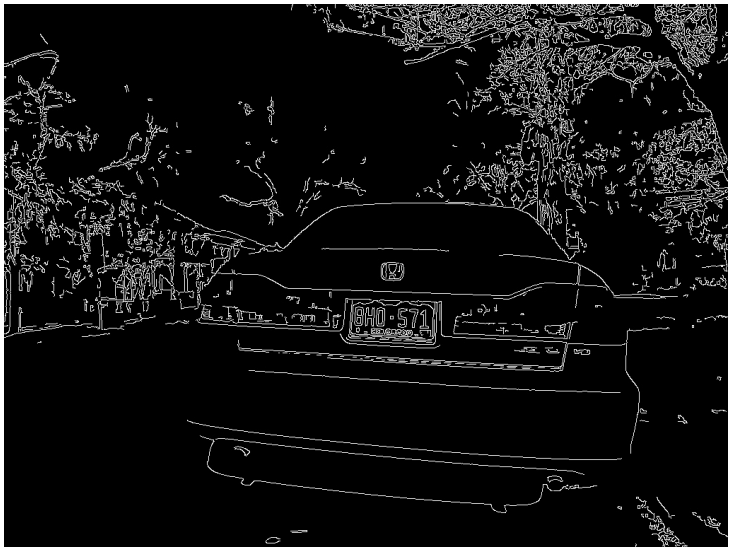

Figure 3: Detected Edges

3. Find contours associated with edges by passing the edges photo (Figure 3) into OpenCV's findContours() function. See Figure 4 for contours found. When finding contours, OpenCV outputs the x & y pixel position of each point on the contour. Therefore, I could easily find the maximum & minimum (x,y) value associated with each contour, providing me with the bounding box enclosing each contour.

Figure 4: Detected Contours

4. Filter Contours to find contours associated with LPCs. As you can see in Figure 4, there are many more contours than just the LPCs. Because I knew some characteristics that were specific to the LPCs, I was able to narrow down which contours most likely were LPC contours. Each of these characteristics I found through testing and are listed below:

    - The width-to-length ratio of the bounding box enclosing each contour. Through inspection, I found this to be between 0.4 & 0.6 for each LPC.
    - The y-position of the bounding boxes top-left corner and the y-position of the bounding boxes bottom-right corner. Because all LPCs will be in a row, they should all have approximately the same starting & ending y pixel location.
    - The width, length, and/or area of the bounding boxes. All LPCs are roughly the same dimensions, so finding bounding boxes with approximately equal widths, lengths, or areas is a good indicated that they are LPCs.

    First, I filtered out all contours that did not have a bounding box that met the width-to-length ratio stated above and saved the minimum & maximum (x,y) values associated with these contours. The algorithm for this can be seen in Algorithm 1 in Appendix A and subsequent image in Figure 5. Next, I used Algorithm 2 in Appendix A to count how many bounding boxes were in similar y-pixel location, length, width and area. Because there are 6 characters in a Colorado LP, I would expect this algorithm to count 5 similar bounding boxes for each LPC bounding box. Next, I wanted to remove all potential LPCs and just find the left & right LPCs, which would give me the bounding box enclosing all LPCs. To do this, I used Algorithm 3 in Appendix A, which compares two potential LPC bounding boxes and makes sure the following is true:

    - The width-to-width ratio of the bounding box enclosing all LPCs to the bounding box enclosing one LPC was between 7 and 10, which was found through

inspection.

- The length-to-length ratio of the bounding box enclosing all LPCs to the bounding box enclosing one LPC was between 0.5 and 2.5, which was found through inspection.
- The two bounding boxes were in similar y-pixel location, length, width and area.
- The bounding box on the left had at least 5 similar bounding boxes to it, which was found in Alrogithm 2 (see Appendix A).

Figure 5: Contours Meeting Width-to-Length Ratio Requirement

Once this was done, I would save the minimum (x,y) value with the bounding box on the left and the maximum (x,y) value with the bounding box on the right, giving me the bounding box enclosing all LPCs. Typically, there was only one candidate left at this point, but more than one was found due to there being over 1000 contours found prior to Algorithm 1 in Appendix A, I would save the bounding box enclosing the potential LPCs with the maximum area. The reason I always chose the maximum area bounding box is because the other candidates were always smaller, probably due to them representing a small group of leaves in a background tree, and this would always give the LPC bounding box.

Now that the bounding box of the LPCs was found, I used OpenCV's rectangle() function to draw a box around the license plate. This can be seen in Figure 6.

Figure 6: Most Probable License Plate Candidate

Issues with Approach

A critical component to building the Vehicle License Plate Blocker is the ability to detect at least three of the four corners of the LP or LPCs. This is because when mapping an image over a target or estimating the targets pose, the plane of the target is needed which requires at least three points. However, given the method detailed above, I was only able to detect two points. Figure 7 shows an angled view of the LP, the desired detected corners (P1 - P4), and the corners that my algorithm is detecting (P1A & P4A). The reason I cannot find P1 - P4 is because the shape of the LPC contour does not always outline the entire character or the character shapes are not straight lines on the left side of the first character & right side of the last character (i.e. The letter "A" is not a straight up & down line on its left or right side. Reference Figure 8). Therefore, I cannot find points P1 - P4, which does not allow me to accurately map a blank license plate to the original or allow me to find the pose of the license plate to the camera.

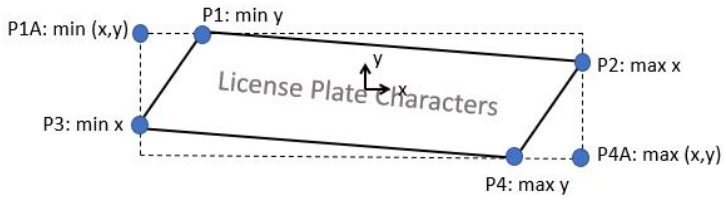

Figure 7: Required LP or LPC Corners for ALPD

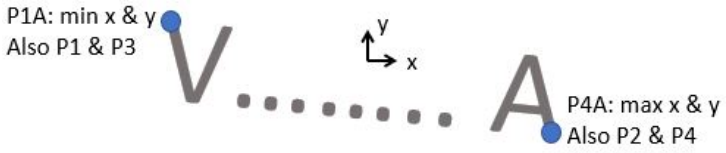

Figure 8: Problem with Certain LPC Shapes

License Plate Masker

Even though I was not able to find the four corner points of the LPCs, I can still use the two detected points found in the previous section to map a mask over the LPCs. To do this, I performed the following steps:

1. Find an image to place over the detected LPCs and choose the four corners that will map to the four corners of the red box shown in Figure 6. I decided to use the image shown in Figure 9.

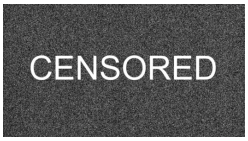

Figure 9: Mask Image

2. Find the Homogeneous Transformation matrix that will warp the mask image corners to the four corners on the LPCs. To do this, I used OpenCV's getPerspectiveTranform() function.

3. Warp masking image to the four corners on the LPCs using OpenCV's warpPerspective() function. This will output a photo that is the same size as the photo shown in Figure 2, with the masked image warped to the same location as the red box in Figure 6 and the rest of the photo being all black pixels.

4. Using three nested for loops, replace the pixel values within the red box in Figure 6 with the pixel values of the masked image in the warped image. The output will be the original image with the mask image covering the license plate, as seen in Figure 10.

Figure 10: Censored License Plate

Vehicle-to-Camera Pose Estimator

Because I cannot find the four corner points of the LPCs, I cannot accurately estimate the pose of the vehicle with respect to the camera. Regardless, I still was able to implement this component using the four corners of the red box shown in Figure 6. To do this, I followed the steps below:

1. In order to accurately estimate the pose of the car with respect to the camera, I first needed to measure the average dimensions of a string of LPCs on a Colorado LP. I found

this to be 10.25" × 2.5" and defined the coordinate axis as the center of the plate. This can be seen in Figure 11.

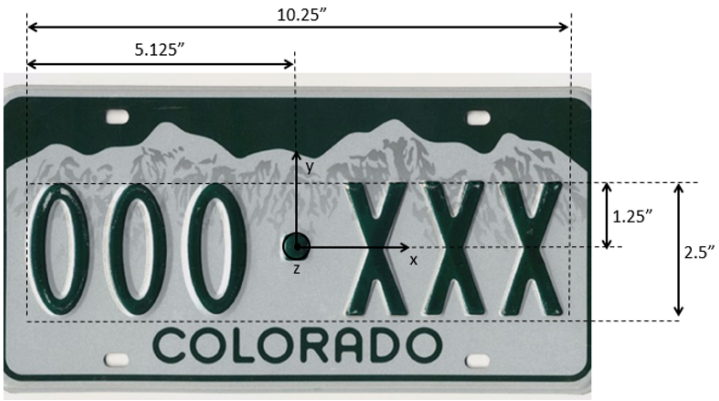

Figure 11: License Plate Dimensions and Coordinates

2. Using 10 images of a checkered board and MATLABs camera calibration tool box, I found my camera's (iPhone XR) intrinsic parameters, which are a focal length of 3058.8 and a pixel width × length of 4032 × 3024.

3. Using the cameras intrinsic parameters, the four corners of the LPCs in the world frame (Figure 11), and the four corners of the LPCs depicted by the red box shown in Figure 6, I solved for the Homogeneous Transformation matrix that projected the LPCs world points to the LPCs camera points using OpenCV's solvePnP() function.

4. Using the Homogeneous Transformation matrix, I projected the points for the world coordinate axis into camera coordinates and using OpenCV's line() function to draw these points on the image. The output will be the original image with the vehicle's coordinate axis drawn over the license plate, as seen in Figure 12.

Figure 12: Vehicle Coordinate Axis

EXPERIMENTS & RESULTS

I tested my LP detection algorithm on about 10 images of vehicles within my neighborhood (all Colorado LPs) and images of vehicles I found on google (non-Colorado LPs). The LPs in these photos were all at different angles and within different lighted environments. Surprisingly, I found my algorithm to work on some non-Colorado LPs, which can be seen in Figure 13, but it wouldn't always detect the first or last LPC, resulting in it not enclosing all LPCs on the LP. As one would expect, I had the best success with detecting Colorado LPs. Also, I

Figure 13: Success on Non-Colorado LPs

found that I typically needed to be within 6 feet of the LP for the detection to work. I did not conduct any tests on LPs at night, but I would not expect the algorithm to have great results

since the LPCs would not be as visible. Overall, I had a successful detecting rate of 70% over the 10 images with about a 5 second processing time. I also attempted to run my algorithm on a 30-fps video, processing every single frame, and was able to detect the LP in about 82% of the frames. However, this took about 30 minutes to process on my personal computer so this algorithm could not perform LP detection in real time. For a sped up demonstration of the LP detection, a demo video can be seen by clicking here. As shown in Figure 7, I was not detecting the correct LPC corners and therefore was not able to accurately map the mask over the LPCs or accurately estimate the vehicle pose with respect to the camera. However, there were not issues with either of these algorithms and should work perfectly if the correct corners were detected.

DISCUSSION

While ALPD has been implemented around the world for over 30 years, it is still the best method for uniquely identifying individual vehicles and research is still being conducted to develop a perfect ALPD system. For this project, I decided to explore the rudimentary technique of detecting the LPCs rather than the LP edges and found it to be quite difficult. As explained above, my algorithm is not robust enough to perform real time LP detection and has issues with detecting the correct LPC corners, resulting in poor LPC blocking and inaccurate vehicle-to-camera post estimation. Additionally, it is restricted to LP detection at a maximum distance of 6 feet from camera to vehicle LP. To fix these issues, I plan on implementing a Sobel edge detector & Hough Transforms, similar to [2][5][6], to detect the correct LP corners. Additionally, I would like to implement a machine learning model to perform LPC recognition. Despite my algorithm's shortfalls, it is still able to detect Colorado LPs in sufficient lighting conditions when parked directly in front or behind vehicles. Overall, I believe my outlined system is sufficient for personal LP detection but needs improvement on LP corner detection for personal LP making and vehicle-to-car pose estimation.

Bibliography

[1] Christos Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Vassilis Loumos, and Eleftherios Kayafas. A license plate-recognition algorithm for intelligent transportation system applications. *IEEE Transactions on Intelligent transportation systems*, 7(3):377–392, 2006.

[2] Christos-Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Ioannis D Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on intelligent transportation systems*, 9(3):377–391, 2008.

[3] Amir Hossein Ashtari, Md Jan Nordin, and Mahmood Fathy. An iranian license plate recognition system based on color features. *IEEE transactions on intelligent transportation systems*, 15(4):1690–1705, 2014.

[4] Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen. Automatic license plate recognition. *IEEE transactions on intelligent transportation systems*, 5(1):42–53, 2004.

[5] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. Automatic license plate recognition (alpr): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology*, 23(2):311–325, 2012.

[6] Tran Duc Duan, TL Hong Du, Tran Vinh Phuoc, and Nguyen Viet Hoang. Building an automatic vehicle license plate recognition system. In *Proc. Int. Conf. Comput. Sci. RIVF*, number 1, pages 59–63, 2005.

[7] Chao Gou, Kunfeng Wang, Yanjie Yao, and Zhengxi Li. Vehicle license plate recognition based on extremal regions and restricted boltzmann machines. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1096–1107, 2015.

[8] Syed Zain Masood, Guang Shu, Afshin Dehghan, and Enrique G Ortiz. License plate detection and recognition using deeply learned convolutional neural networks. *arXiv preprint arXiv:1703.07330*, 2017.

[9] Muhammad Sarfraz, Mohammed Jameel Ahmed, and Syed A Ghazi. Saudi arabian license plate recognition system. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pages 36–41. IEEE, 2003.

[10] Ying Wen, Yue Lu, Jingqi Yan, Zhenyu Zhou, Karen M von Deneen, and Pengfei Shi. An algorithm for license plate recognition applied to intelligent transportation system. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):830–845, 2011.

Appendix A: Algorithm Pseudocode

Algorithm 1 Find Contours Meeting LPC Width-to-Length Ratio

```
Find contours in image using OpenCV's findContours()
for Each contour in list of contours do
    x_min = 10000 ← Define dummy variable larger than any pixel value for finding minimum

    y_min = 10000 ← Define dummy variable larger than any pixel value for finding minimum

    x_max = 0 ← Define dummy variable smaller than any pixel value for finding maximum
    y_max = 0 ← Define dummy variable smaller than any pixel value for finding maximum
    for Each (x,y) position in contour do
        if x > x_max then
            x_max = x
        end if
        if y > y_max then
            y_max = y
        end if
        if x < x_min then
            x_min = x
        end if
        if y < y_min then
            y_min = y
        end if
    end for

    Now we have the max & min (x,y) values associated with each contour
    if y_min ≠ y_max then
        Width-to-Length Ratio= (x_max − x_min) / (y_max − y_min)
    end if
    if 0.4 < Width-to-Length Ratio < 0.6 then
        Potential LPC Contour = [x_min, y_min, x_max, y_max]
    end if
end for
```

Algorithm 2 For a specific bounding box, count bounding boxes that are similar to it (LPCs will all have similar bounding boxes)

for Each saved max & min (x,y) associated with a potential LPC contour i **do**
  $w_i = x_{i,max} - x_{i,min} \leftarrow$ Contour i Width
  $h_i = y_{i,max} - y_{i,min} \leftarrow$ Contour i Length
  $a_i = w_i h_i \leftarrow$ Contour i Area
  for Each saved max & min (x,y) associated with a potential LPC contour j **do**
    $w_j = x_{j,max} - x_{j,min} \leftarrow$ Contour j Width
    $h_j = y_{j,max} - y_{j,min} \leftarrow$ Contour j Length
    $a_j = w_j h_j \leftarrow$ Contour j Area

    Now, check if contour i and j min y-positions are close to verify they are in a row
    if $y_{i,min} < y_{j,min} + \frac{h_j}{1.5}$ and $y_{i,min} > y_{j,min} - \frac{h_j}{1.5}$ **then**
      Check if contour i and j max y-positions are close to verify they are in a row
      if $y_{i,max} < y_{j,max} + \frac{h_j}{1.5}$ and $y_{i,max} > y_{j,max} - \frac{h_j}{1.5}$ **then**
        Check if i and j widths are close in size
        if $w_i < w_j + \frac{w_j}{1.5}$ and $w_i > w_j - \frac{w_j}{1.5}$ **then**
          Check if i and j areas are close in size
          if $a_i < a_j + \frac{a_j}{3}$ and $a_i > a_j - \frac{a_j}{3}$ **then**
            Counter = Counter + 1 $\leftarrow$ Count number of contours j's that meet these requirements for contour i
          end if
        end if
      end if
    end if
  end for
  Save counter number for contour i
end for

Algorithm 3 Find most likely LP Candidates

for Each saved max & min (x,y) associated with a potential LPC contour i **do**
  $w_i = x_{i,max} - x_{i,min} \leftarrow$ Contour i Width
  $h_i = y_{i,max} - y_{i,min} \leftarrow$ Contour i Length
  $a_i = w_i h_i \leftarrow$ Contour i Area
  for Each saved max & min (x,y) associated with a potential LPC contour j **do**
    $w_j = x_{j,max} - x_{j,min} \leftarrow$ Contour j Width
    $h_j = y_{j,max} - y_{j,min} \leftarrow$ Contour j Length
    $a_j = w_j h_j \leftarrow$ Contour j Area
    if $w_i \neq w_j$ and $h_i \neq h_j$ **then**
      $W_{LPC} = x_{j,max} - x_{i,min} \leftarrow$ Width of bounding box (bb) enclosing all LPCs
      $W_{ratio} = \frac{w_i}{W_{LPC}} \leftarrow$ Width ratio of bb enclosing all LPCs & bb enclosing LPC i
      $H_{LPC} = y_{j,max} - y_{i,min} \leftarrow$ Length of bb enclosing all LPCs
      $H_{ratio} = \frac{h_i}{H_{LPC}} \leftarrow$ Length ratio of bb enclosing all LPCs & bb enclosing LPC i
    end if

    Verify the width & length ratio between the LPCs combined and a single LPC falls within inspected ratios
    if $7 < W_{ratio} < 10$ and $0.5 < H_{ratio} < 2.5$ **then**
      Verify LPC i and j meet all requirements in Algorithm 2
      if $y_{i,min} < y_{j,min} + \frac{h_j}{1.5}$ and $y_{i,min} > y_{j,min} - \frac{h_j}{1.5}$ **then**
        if $y_{i,max} < y_{j,max} + \frac{h_j}{1.5}$ and $y_{i,max} > y_{j,max} - \frac{h_j}{1.5}$ **then**
          if $w_i < w_j + \frac{w_j}{1.5}$ and $w_i > w_j - \frac{w_j}{1.5}$ **then**
            if $a_i < a_j + \frac{a_j}{3}$ and $a_i > a_j - \frac{a_j}{3}$ **then**
              Make sure at least 5 bounding boxes are in a row (LP has 6 LPCs)
              if Counter > 5 **then**
                LP has likely been found. If comparing contours i and j have gotten this far through the filters above, then contour i is likely the left LPC and contour j is likely the right LPC. Save min (x,y) values for contour i and max (x,y) values for contour j.
              end if
            end if
          end if
        end if
      end if
    end if
  end for
  Save counter number for contour i
end for