Lane Line Finding Project Final Report

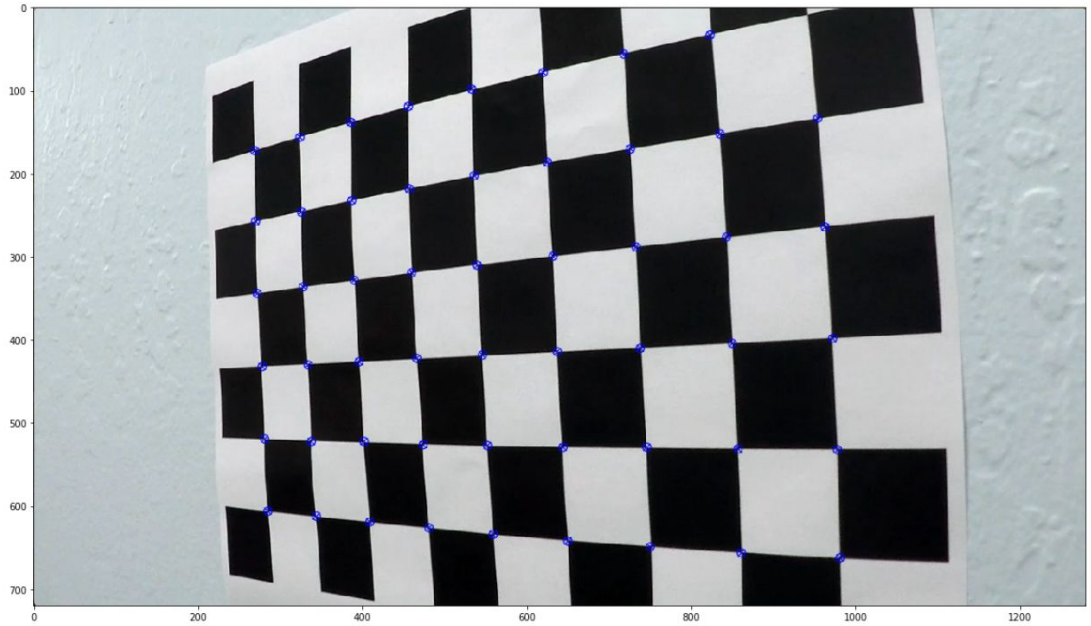
CSC 381/481: Image Processing
Cody Nicholson
Merga Tafa

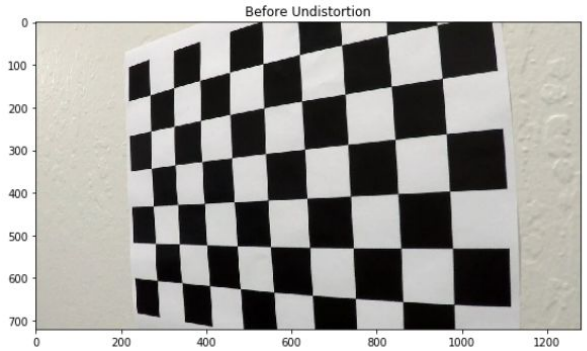
Abstract:

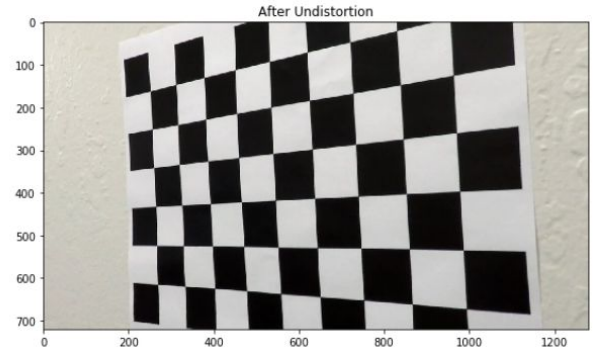
We created a program to detect lane lines in front of a moving vehicle. We did this by correcting distortion caused by the camera lens, using color transformations and gradients to create a thresholded binary image, applying a perspective transformation to turn our image into a top-down view, detecting lane pixels and fitting them to find the lane boundary, determining the curvature of the lane and vehicle position with respect to center, warping the detected lane boundaries back onto the original image, and outputting a visual display of the lane boundaries with the numerical estimation of lane curvature and vehicle position included in the image.

Methodology Step 1 - Fix distortion caused by camera lens in images:

The round glass of the lens of the camera used to take the images in our dataset caused distortion. Our first step of our methodology was to eliminate this distortion using some of the tools built into computer vision. By using the same camera used to create our dataset to take images of a chessboard, we can use the image of the chessboard as a reference to undistort the images.







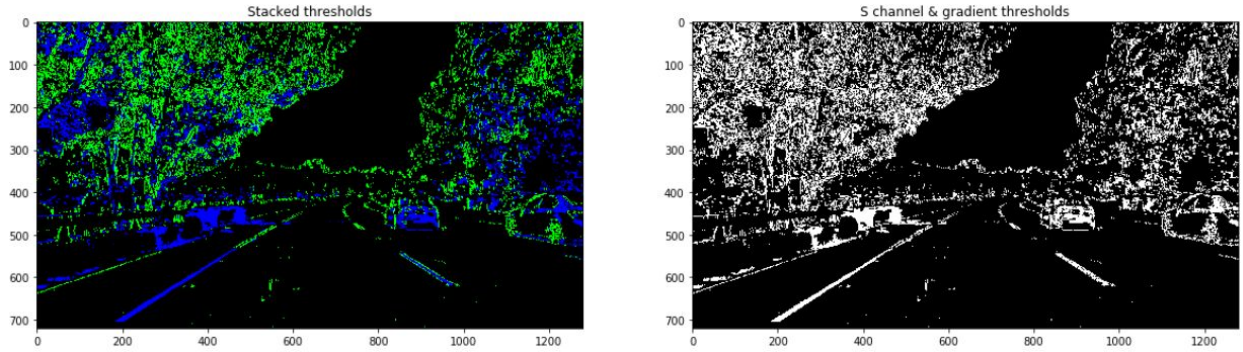
As you can see in the top image, we have detected all of the corners of each square on the chessboard. Since we know there should be a straight line between each corner and the adjacent corners, we can use all these straight line references to undistort the image. You can see the difference in the two images below the large chessboard image.

We were able to accomplish this using the Python library known as Computer Vision. The library has functions defined within it to detect the chessboard corners and undistort the image.

Methodology Step 2 - Use color transforms and gradients to create a thresholded binary image:

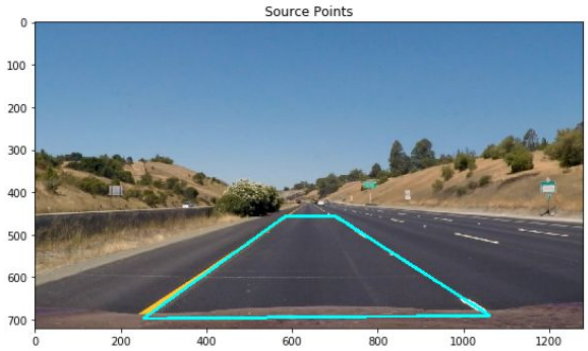
The second step of our methodology was to threshold our images so that we could decide which color channel would define our lane lines the best. After segmenting the different color channels, we stacked them on top of each other to see which color channel would best capture the lane lines. We labeled the color channel that did the best ‘s’.

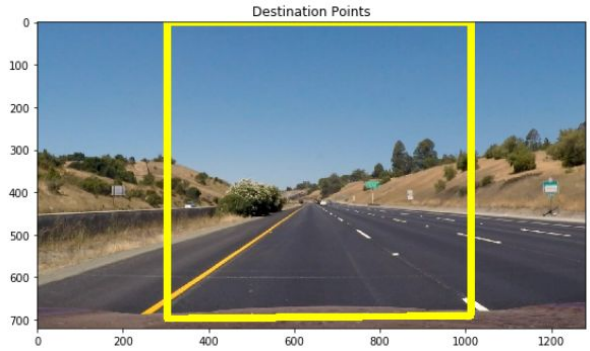
Once we had selected the ‘s’ color channel, we then applied the Sobel X filter so that we could detect the brightest pixels in the vertical lane lines. Then we detected the gradient of the image so we could make our lines even more accurate. You can see our output below:



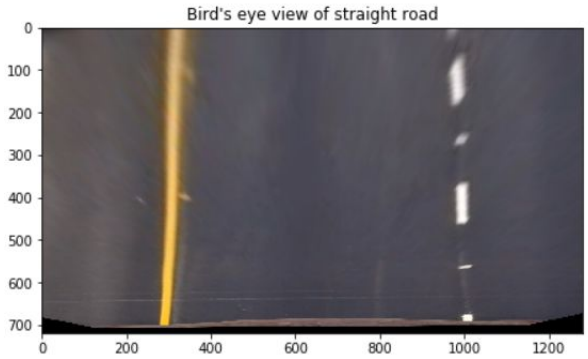
Methodology Step 3 - Apply a perspective transform to the binary image to change it to top-down view:

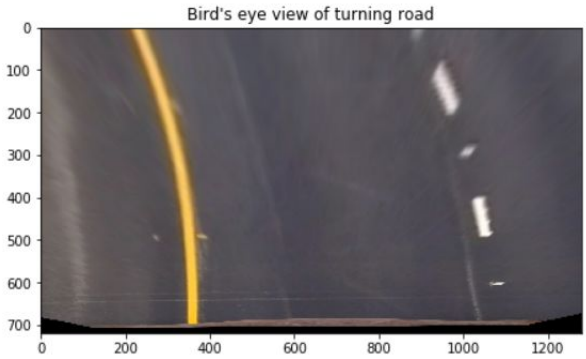
The third step was the apply a perspective transform to our binary images to eliminate noise and increase the accuracy of our algorithm. The first part of this step was to define our source and destination points. The source points would define a polygon that represents roughly where the lane is located in the image. The destination points define another larger polygon that represents where we would like to stretch the lane captured by the source points to:





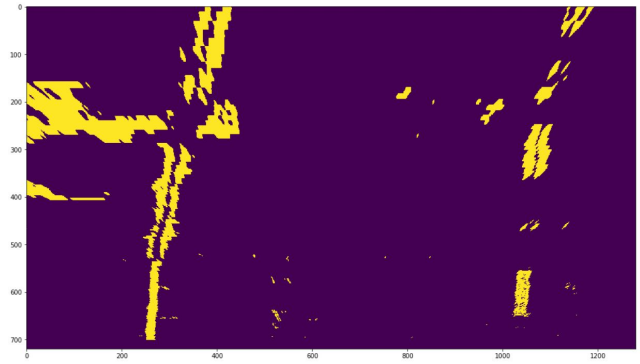
We hard coded the source points since we know that the road will be approximately in the same place in all of our test images. The next step was to stretch the source point image into the destination image. To do this we used some methods included in the computer vision library. This was our result:

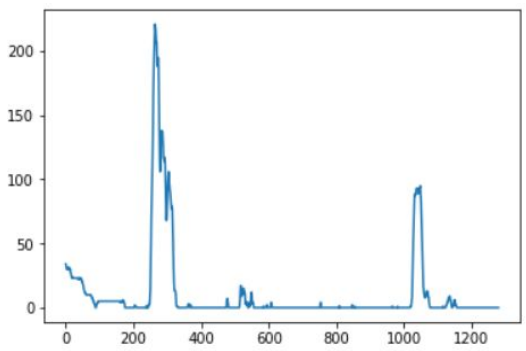




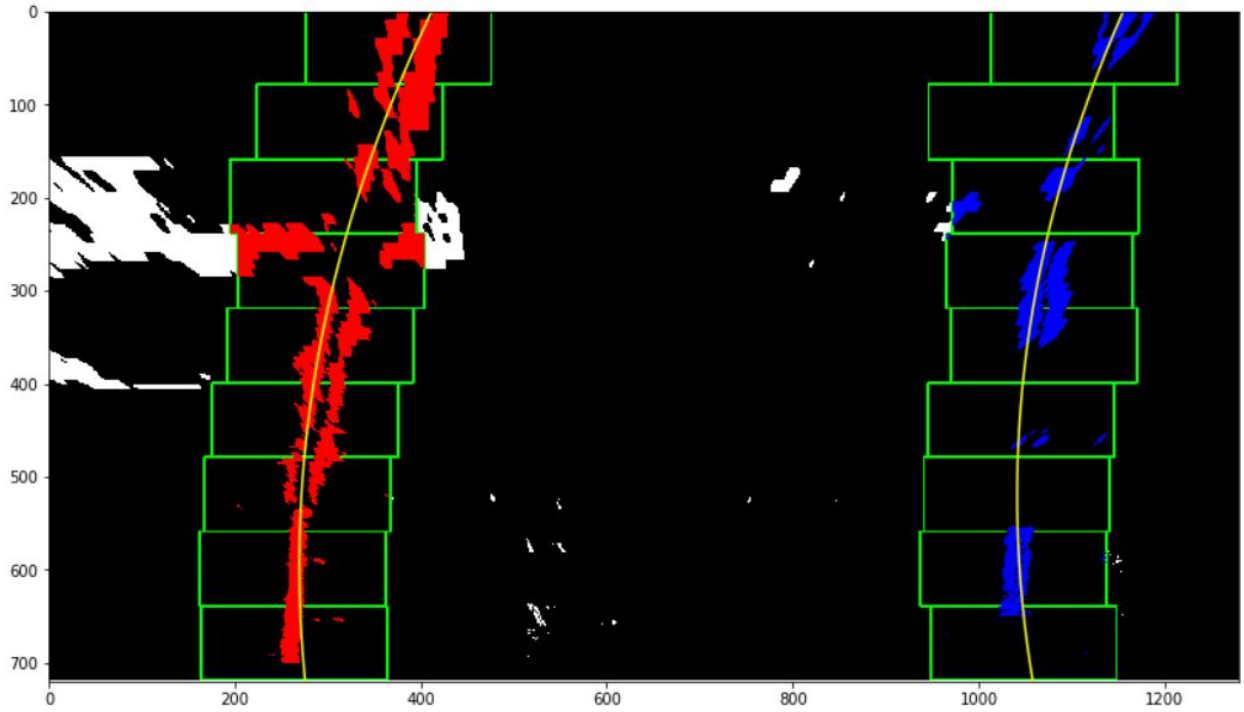
Methodology Step 4 - Detect lane pixels and fit them to find the lane boundary:

Our fourth step was to use histogram equalization to find the lane boundaries. We did this by creating a histogram that represented the locations of the brightest pixels in our binary image, as seen below:





Once we had identified where the lane lines were in our binary image, we used the sliding window approach to parse through the binary image to begin to understand exactly where the complete lane lines were located with higher accuracy. We implemented the sliding window method by first identifying the two peaks in our histogram, choosing to parse the image from left to right using nine windows, identifying the x and y coordinates of all non-zero pixels in the image, creating two lists to hold pixels from each of the two lane lines, centering our windows over the lane line, converting the distances between pixels into meters, fitting the pixels to the new real-world space, and calculating the intercept points of the left and right lane lines. After we had calculated the intercept points and fit the pixels into a real-world space in meters, we were able to determine the vehicle's deviation from the center of the lane in our test images. Since we centered our windows over the lane line and we knew the locations of all the pixels in each lane line, we were able to approximate the curvature of the lane line based on how the windows are staggered over the lane line. We drew a thin yellow line over the lane line representing where our program perceives the lane line to be exactly:



This was the most critical step of the process since we are no longer preprocessing, we are now actually implementing the algorithm used to find the lane lines. Now that this step is complete, we no longer need to look at the binary images anymore.

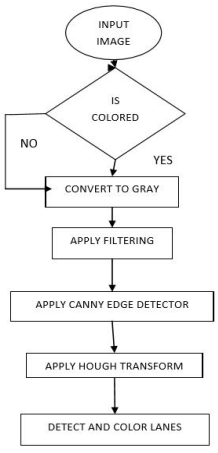
Methodology Step 5 - Output visual display of the lane boundaries:

Finally, in our fifth step we take all the information we received from our previous steps and use it to draw the lane lines on top of the actual test image as seen below:

We output some of the details found in our steps in the top left corner of the image. The “Left” and “Right” variables tell us about the curvature of the lane. Since the “Left” value is slightly greater than the “Right” value in the above image, we know that we are turning right. We were able to determine this based on the sliding window method we used in Step 4 to put all the pixels in each lane line into a separate array and then find the intercept points.

Reference 1 - Kaur, Gurveen. "Lane Detection Techniques: A Review.":

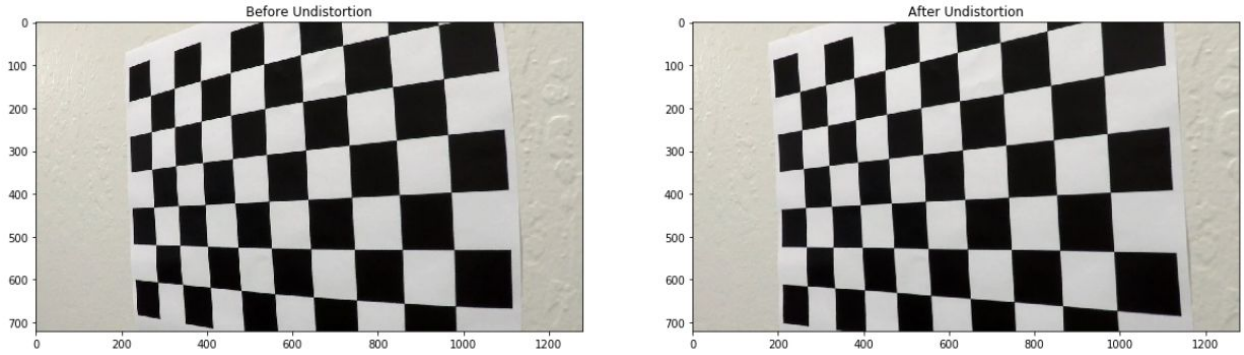
We reviewed the paper "Lane Detection Techniques: A Review." written in 2015 by Gurveen Kaur so that we could get an idea of how to design our project. As you can see in the image on the right, Gurveen decided to use the Canny Edge detection algorithm with the Hough Transformation algorithm. We experimented with this design, but found that it had many limitations.



The most notable limitation was that it could not process a curved line since the Hough Transformation was used to create straight lines. Another limitation was that the Canny algorithm picks up a lot of noise, which makes it very inaccurate. All that said, we did learn a lot from this paper but we did not apply many of the same techniques in our project.

Reference 2 - Weng, Juyang. "Camera Calibration with Distortion Models and Accuracy of Evaluation."

The image dataset that we used for our project was slightly distorted by the camera lens used to take the pictures. To resolve this issue, we reviewed the paper "Camera Calibration with Distortion Models and Accuracy of Evaluation" written in 1992 by Juyang Weng. The paper taught us how to undistort the images by using a chessboard image taken with the same camera as a reference.



The first step of this process is to detect all of the corners of the 64 squares on the chessboard. Since we know that there should be a straight line between each corner and its adjacent corners, we can use the corners as references to undistort the image. See the above image that shows the distorted image on the left, and the undistorted image on the right.

Reference 3 - Eynard, D. "Structure-preserving color transformations using Laplacian commutativity."

For our third paper, we read "Structure-preserving color transformations using Laplacian commutativity." by D. Eynard. We read this paper since we were considering using a Laplacian filter for our lane line edge detection and segmentation. We were informed again that the Laplacian filter can detect edges in all directions. After carefully reviewing the paper, we decided that the Sobel X filter would be a much better choice since the Laplacian filter struggles with noise. Our test images have lots of noise that could easily throw off the Laplacian filter's edge detection.

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

1	-2	1
-2	4	-2
1	-2	1

Reference 4 - Mohammad, Elham. "Study Sobel Edge Detection Effect on the Image Edges Using MATLAB."

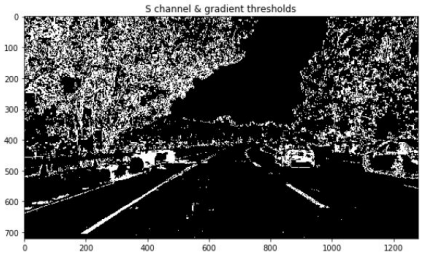
After deciding not to use the Laplacian filter, we read the paper "Study Sobel Edge Detection Effect on the Image Edges Using MATLAB." written in 2014 by Elham Mohammad. The paper discussed the two configurations for the Sobel filter: X and Y. Sobel X can be used to detect bright vertical lines, and Y can be used to detect bright horizontal lines. We decided to use Sobel X since the lane lines appear as vertical in our image dataset.

-1	0	+1
-2	0	+2
-1	0	+1

x filter

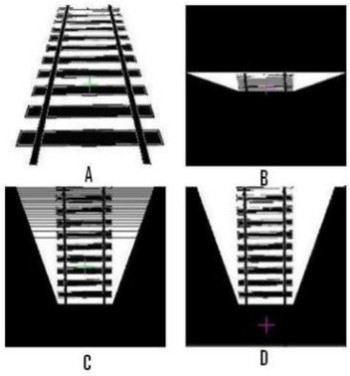
+1	+2	+1
0	0	0
-1	-2	-1

y filter



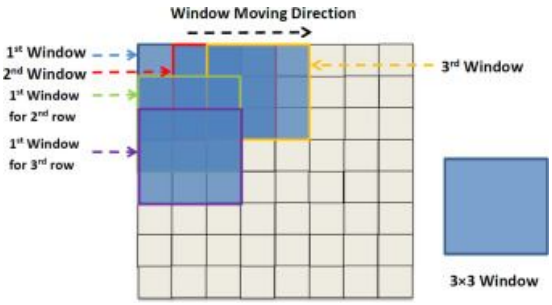
Reference 5 - Venkatesh, V. "Transformation Techniques."

The fifth paper we chose to review was titled "*Transformation Techniques*" written in 2012 by V. Venkatesh. It discussed how to transform our images from a forward-facing perspective to a top-down perspective. This was one of the most important techniques for ensuring high accuracy since we remove much of the noise in this process. As you can see in the image on the right, the original image is transformed so that it appears to be in a top-down view. Notice that in the top-down view, all the lines are straight and easy to detect. This is great for our purposes since we want our program to have an easy time processing the image.



Reference 6 - Lee, Han-Wook. "Sliding window approach based Text Binarization from Complex Textual images."

The last paper we reviewed was titled "*Sliding window approach based Text Binarization from Complex Textual images*." written in 2010 by Han-Wook Lee. This paper discussed how to utilize the sliding window method to parse through an image to detect all of the pixels of interest. We used this technique to select the lane line pixels so that we could calculate where the vehicle was located relative to the center of the lane.



Evaluation:

Our algorithm performed surprisingly well given all the noise in our dataset. We were able to clearly identify the lane lines, and highlight the lane in front of the vehicle. Unfortunately, this was hard for us to quantify because we found it hard to define what 100% accuracy was since we did so much approximating in our implementation. We did find a few limitations of our algorithm.

As you can see in the below image, our program has made the lane too wide on the left side. This is likely because the bright concrete road has thrown off our thresholding since there are so many bright pixels. If we were to continue working on this project, we might consider keeping the color of the images (not using grayscale) so that we can easily segment out the bright gray/tan concrete without removing the lane lines.

Division of Labor 1 - Cody:

I was responsible for all of the programming that went into our project. I took all the information that Merga obtained from his searches and found ways to integrate it into our project by researching how to use Python libraries to recreate what was done in the papers. I mentored Merga on how to implement various image processing techniques in Python.

In addition to programming, I helped with the organization and formatting of both our paper and our presentation. I also verified that everything was worded correctly and concisely so that we could have the most in-depth description of our project in a reasonable amount of time. If you would like to see our code, you can view it by going to our project website at: https://codynicholson.github.io/Lane_Line_Finding_Project/.

Division of Labor 2 - Merga:

I gathered many different references to use in our project. I reviewed these references and created a slide for each reference in our final presentation, and a paragraph for each reference in our final paper. While I was unable to help much with the programming side of the project, I did learn what each line of code did in our project.

Conclusion:

We learned a lot in completing this project. We got to experiment with computer vision, become more familiar with linear algebra, become better at programming in python, and practice many of the techniques we learned in class. While our implementation may not be perfect quite yet, given more time we are sure that we could optimize our project to provide even better results. We hope that our research leads to a safer future for our roadways.

Works Cited:

1. Kaur, Gurveen. "Lane Detection Techniques: A Review." DAVIET College, Jalandhar (Pb.) India, 10 February 2015, http://research.ijcaonline.org/volume112/number10/pxc3900923.pdf.
2. Weng, Juyang. "Camera Calibration with Distortion Models and Accuracy of Evaluation." Institute of Electrical and Electronics Engineers , 10 October 1992, http://ieeexplore.ieee.org/document/159901/.
3. Eynard, D. "Structure-preserving color transformations using Laplacian commutativity." University of Lugano, Switzerland, 1 November 2013, https://arxiv.org/pdf/1311.0119.pdf/.
4. Mohammad, Elham. "Study Sobel Edge Detection Effect on the Image Edges Using MATLAB." International Journal of Innovative Research in Science, India, March 2014, https://www.ijirset.com/upload/2014/march/68_Study.pdf/.
5. Venkatesh, V. "Transformation Techniques." International Journal of Scientific & Engineering Research, 5 May 2012, https://github.com/CodyNicholson/Lane_Line_Finding_Project/raw/master/pdfs/Simple_Birds_Eye_View_Transformation_Technique.pdf/.
6. Lee, Han-Wook. "Sliding window approach based Text Binarization from Complex Textual images." International Journal on Computer Science and Engineering, IJCSE, 18 March 2010, https://arxiv.org/abs/1003.3654.