Project 2

Perception for Autonomous Robots

March 24, 2022

Instructors:
S. Charifa

Student:
Jerry Pittman, Jr. (117707120)

Semester:
Spring 2022

Course code:
ENPM673

Contents

List of Figures

1 Generated Videos

Videos generated for the project are located here and are also hyperlinked in their appropriate sections.

2 Problem 1: Histogram Equalization

In order to make the histogram equalization iterative, I first compiled the 25 images into a movie. I also resized the images/movie so that it would be divisible by 8 by making it 480x640 (lowest resolution that was closest to original resolution of 370x1224). This took a lot of trial in error in determining a good value of frames per second (fps) and also fixing the ordering of the videos using lamdba function so less glitchy. I ended up going with 3 fps and to ensure follow-on videos made matched this fps value.

For conducting the histogram equalization, I converted the image frame into HSV then used the V values for creating a histogram. I then ran this histogram through a Cumulative Distribution Function (CDF)to normalize the pixels.

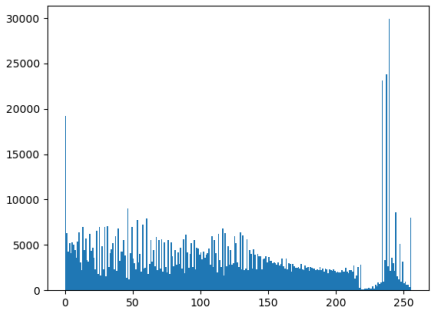

Figure 1: Frame 2 Histogram.

I then compared the results of applying the CDF results to the original image and was happy with the results of improving the contrast of the image and movie.

Figure 2: Frame 2 Comparison of original image to Histogram Equalization Image

For conducting Contrast Limiting Adaptive Histogram Equalization (CLAHE), I originally tried to reuse functions from regular histogram equalization but determined that only my histogram generating figure functions (which only call on matplotlib functions) will be able to be reused. This was determined after reading [8], [10]–[12] and looking at different methodologies have been performed by others. For CLAHE I used this procedure:

Algorithm 1 CLAHE Functions

function conductAdaptiveHistogramEqualization(image)
    Set kernel size by dividing image into eighths
    Pad image to ensure each dimension is multiple of kernel size and has a buffer so don't run out of array size
    Create Look Up Table to convert dynamic range of image to desired output range. Cumulative sum of each histogram bin.
    Map each kernel "block" based on grayscale values
    Create histogram
    Clip histogram using *clipHistogram* function
    Conduct bi-linear interpolation to remove artifacts
    Combine image blocks together and remove padding
end function
function clipHistogram(hist, clip_limit)
    Determine excess (amount of histogram above clipLimit)
    Clip Histogram above clip limit and evenly distribute excess into each bin while ensuring individual bins do not exceed clipLimit.
end function

This part of Problem 1 was very difficult for me. I think that either my grayscale limits value, clipLimit, or perhaps my mapping and breaking up the image into small enough kernels but I couldn't get my CLAHE image to look better than original or histogram equalized. It either comes out looking darker overall or develops technicolor patterns and makes the image discernible. I also went back and forth between conducting this analysis in HSV like I did earlier or to do CLAHE in grayscale or RGB. I ended up keeping the video/image in RGB.

Resulting video is here. I determined my results were good enough since image wasn't overly bright or dark and would come back to this problem after I completed problems 2 and 3.

I believe that this pipeline will work on other videos but would need to verify/update the frames per second and resolution of output video to match the input video/images.

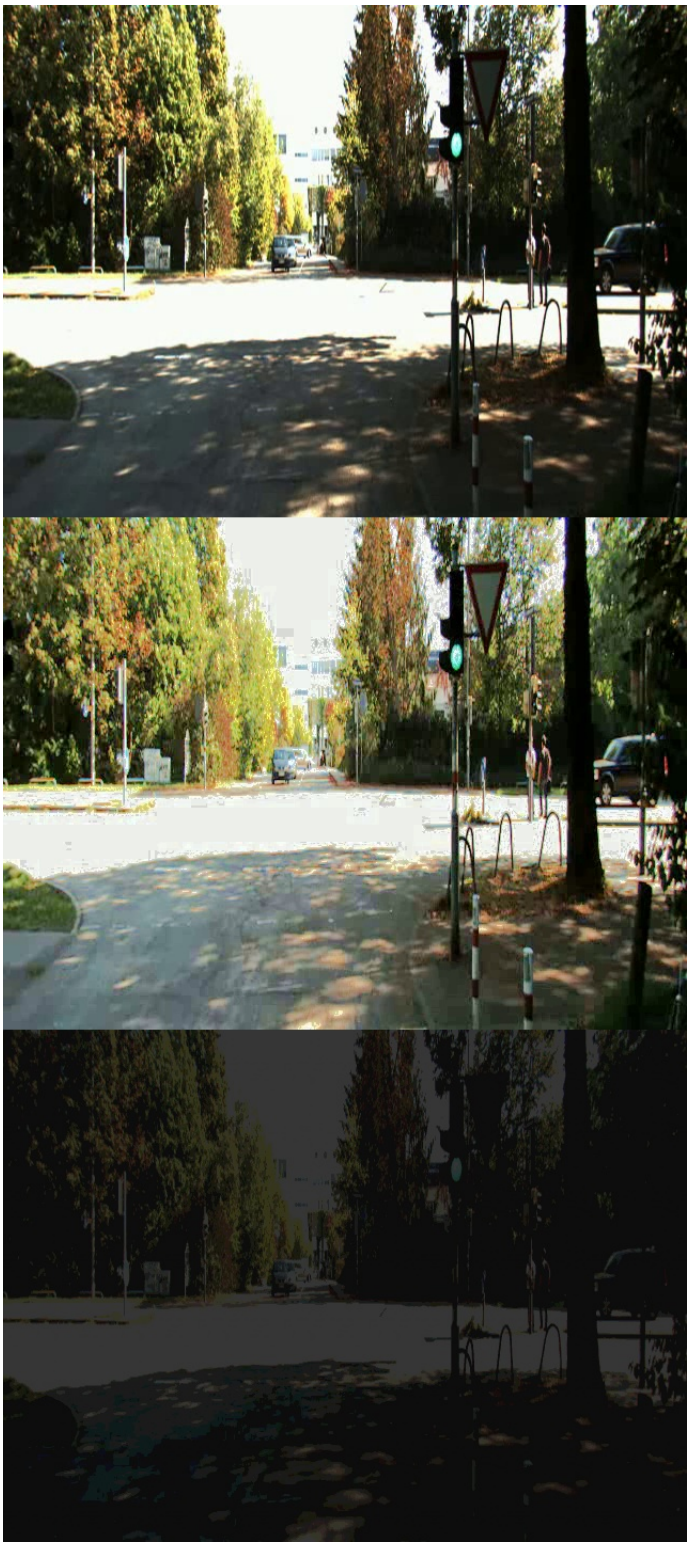

Figure 3: Frame 2 Comparison of original image to Histogram Equalization Image and CLAHE

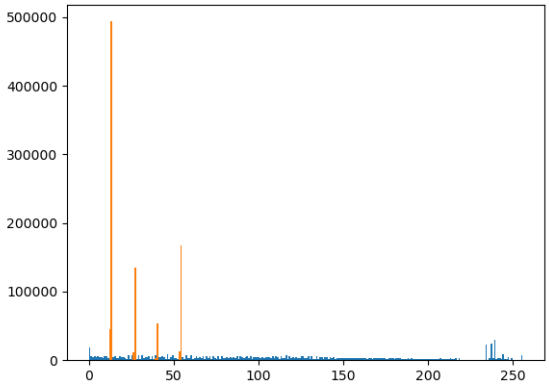

Figure 4: Frame 2 Histogram after CLAHE clipping

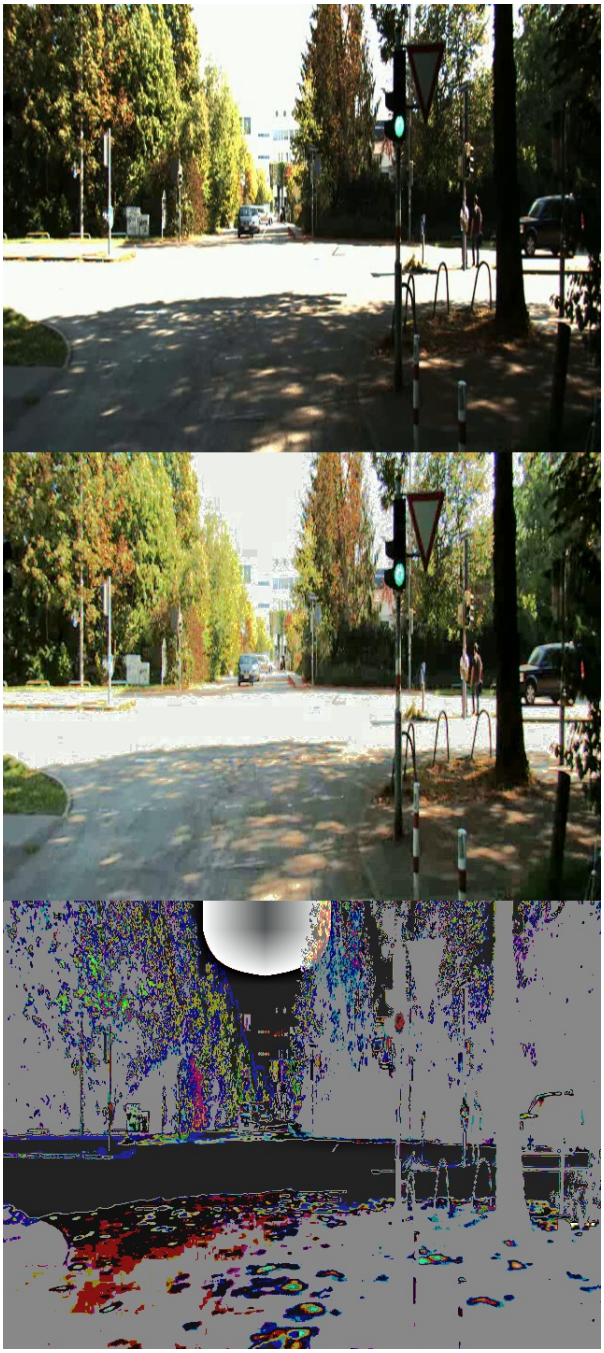

Figure 5: Frame 2 Comparison with clipLimit=40

3 Problem 2: Straight Lane Detection

I used the following pipeline for detecting the lanes and coloring them.

Algorithm 2 Straight Lane Detection Pipeline

1: **function** Detect and Color lanes(image)
2: Convert frame to Grayscale
3: Reduce noise using medianBlur; kernel size of 5.
4: Use Canny for edge detection
5: Use Probablistic Hough Transform for finding lines/lanes.
6: Draw lines on canny resulting image (converted back to BGR so colored lines show up as colors) and original video.
7: **end function**

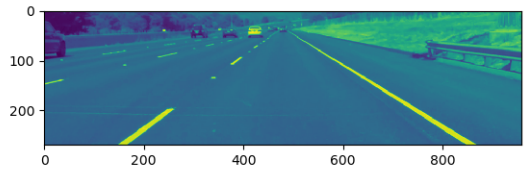

Figure 6: Cropped Frame 2 with just region of interest

I did gaussianBlur initially but found medianBlur gave better results by allowing me to dictate kernel size.

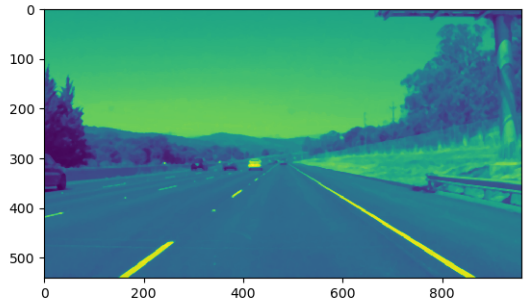

Figure 7: Frame 2 with medianBlur

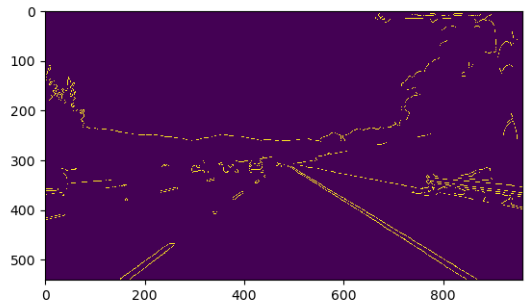

Figure 8: Frame 2 Edges

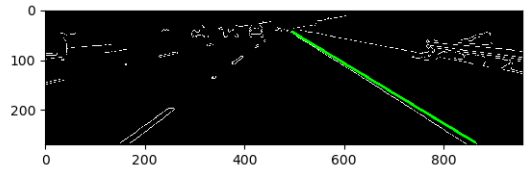

Figure 9: Frame 2 Edges with line just solid line

I had lots of difficulty in tagging dashed (lane lines) than the solid (direction lines). There was a good amount of fine tuning to get the best marking of lines with minimal overlap but the "red" dashed lines still overlaps part of "green" solid lines since the openCV Hough Lines functions do not seemt to have a maxLineLength attribute. I tried making an iterative loop for removing duplicates from the dashed array from those previously found in the solid lines array with no joy. I also did some attempts with clipping the video frames to just the region of interest (the road) but then that added complications in drawing those lines onto the original video/images.

My understanding of how Hough Lines work is it converts pixel coordinates to Hough plane coordinates (r and θ). It then determines the occurrence to clip the infinite lines. You can also control what the max gap is between lines to separate or count lines as the same, threshold, number of lines possible, angle resolution, distance resolution, and minimum length of lines. Basically, points in image space are lines are Hough space and points in Hough space are lines in image space.

I believe that this pipeline will work on other videos but would need to verify/update the frames per second and resolution of output video to match the input video/images.

Resulting video with lanes colored is here. There can be some further fine-tuning of hough transform parameters and some process to ensure that if a line is "solid" then it does not get marked as

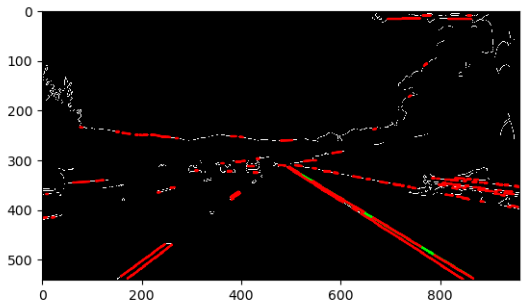

Figure 10: Frame 2 with lines drawn on pre-processed image

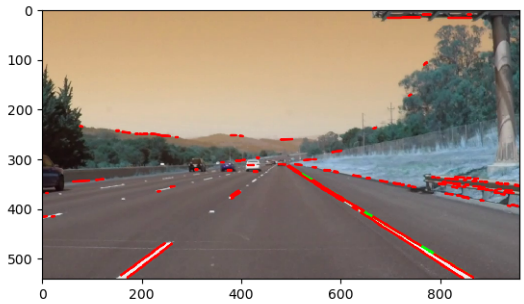

Figure 11: Frame 2 with lines drawn on original image

"dashed" so not competing colors for the solid lines.

4 Problem 3: Predict Turn

I used the following pipeline for turn prediction:

Algorithm 3 Turn Detection Pipeline

function Detect Lane curvature and plot polygon(image)
    Reduce noise using medianBlur; kernel size of 5.
    Crop the image into halves to focus on portion of video/frame that has lanes contained
    Find Homography to warp image
    Convert image to HLS in order to use intensities of lane colors
    Create masks for yellow and white lane markings
    Solve for histogram of intensities to find where right and left lanes are based on their colors.
    Plot polygon, green, on video.
    Create mask for road for unwarping the image.
    Re-stitch upper (unedited) and lower halfs (edited with polygons) of image.
    Solve for radius curvature using right lane:

$$RadiusOfCurvature = \frac{(1 + (2 * Coeff_{right_0} * points_{right_1}) + Coeff^2_{right_1})^{3/2}}{2 * Coeff_{right_0}}$$

Where $right_0$ is present/bottom of video and $right_1$ is future lane location.
    Add text to video indication if lane is turning "left", "straight", or "right" and it's associated curvature radius.
end function

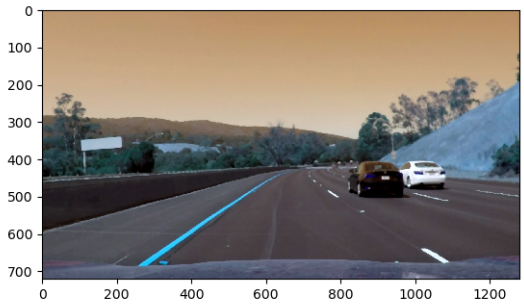

Figure 12: Median Blur applied

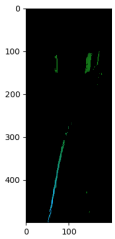

Figure 13: Yellow Mask

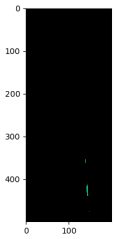

Figure 14: White Mask

I had issues in figuring out how to solve for and plot the polygon indicating the lane, solving for the curvature of the lane (turn prediction), and masks based on lane color (yellow vs white) and used portions of [6] in order to execute these tasks.

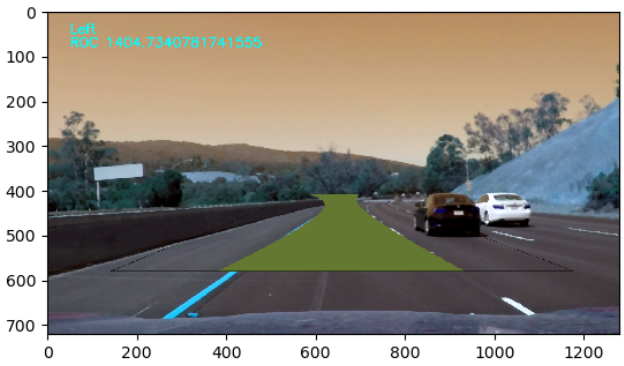

Figure 15: Resultant Video Frame

Code is reusable and user would just need to modify the sizing of the image and name of input image.

Homography means to me is a way of comparing points in two "worlds" so that you can warp or unwarp an image to change the perspective (bird's eye view in my case for determining masks and lane curvature) and then unwarp to look like original image.

Resulting video with lane colored and lane curvature value and Left/Right/Straight displayed on video frame is here.

5 Resources

5.1 My Github repository

https://github.com/jpittma1/ENPM673-Project2-Lane_Detection.git

5.2 References

Whole bibliography

[1] A. Aggarwal. "Lane$_d$*etection*." (), [Online]. Available: https://github.com/arp95/lane_detection.git. (accessed: 04.11.2022).

[2] CalTech. "Line detection by hough transformation." (), [Online]. Available: https://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf. (accessed: 04.21.2022).

[3] S. Charifa, "Enpm673 spring 2022 notes," University of Maryland - College Park, MAGE, College Park, MD, Tech. Rep. Lectures, Jan. 2022.

[4] Gokul. "Lane detection in python." (), [Online]. Available: https://github.com/h-gokul/LaneDetection.git. (accessed: 04.11.2022).

[5] A. Jadhav. "Lane-detection-and-turn-prediction." (), [Online]. Available: https://github.com/iamjadhav/lane-detection-and-turn-prediction.git. (accessed: 04.11.2022).

[6] ——, "Lane-detection-and-turn-prediction." (), [Online]. Available: https://github.com/iamjadhav/lane-detection-and-turn-prediction/blob/main/Problem_2_DataSet_2.py. (accessed: 04.24.2022).

[7] G. A. Kumar. "Enpm673 2020." (), [Online]. Available: https://github.com/govindak-umd/ENPM673.git. (accessed: 04.11.2022).

[8] K. Lucknavalai and J. P. Schulze. "Real-time contrast enhancement for 3d medical images using histogram equalization." (), [Online]. Available: http://web.eng.ucsd.edu/~jschulze/publications/Lucknavalai2020.pdf. (accessed: 04.11.2022).

[9] openCV. "Feature detection: Hough transformation." (), [Online]. Available: https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb. (accessed: 04.21.2022).

[10] ——, "Histograms - 2: Histogram equalization." (), [Online]. Available: https://docs.opencv.org/3.4/d5/daf/tutorial_py_histogram_equalization.html. (accessed: 04.11.2022).

[11] stackOverflow. "Histograms - 2: Histogram equalizationdepth error in 2d image with opencv python." (), [Online]. Available: https://stackoverflow.com/questions/19103933/depth-error-in-2d-image-with-opencv-python. (accessed: 04.15.2022).

[12] wikipedia. "Adaptive histogram equalization." (), [Online]. Available: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Efficient_computation_by_interpolation. (accessed: 04.15.2022).