Lane Detection Project

Aviad Kariv

VIDEO LINKS:

LANE DETECTION HIGHWAY DAYLIGHT

https://www.youtube.com/watch?v=X6ND38tKoSM

NIGHTTIME PROXIMITY

https://www.youtube.com/watch?v=o2C_oNhlK5U

NIGHTTIME CROSSWALK

https://www.youtube.com/watch?v=GSAxxWnDdlc

Part 1 – Lane Detection:

The roadmap to detect lanes was as follows:

1. Pre-process the image, so that the lanes will be easily visible, and will have minimal noise
2. Detect and track the lanes when the road is straight
3. Enabling the ability to change lanes

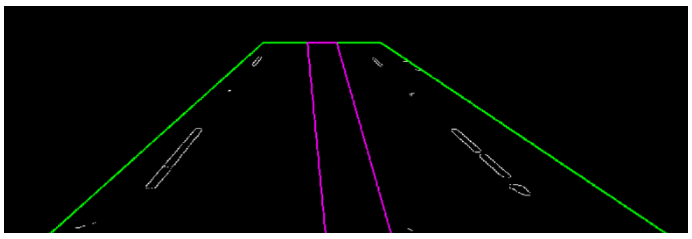
These steps were contained in two files – lane_preprocessor.py and lane_detector.py.

Preprocessing took many trials and errors but the idea was always to make it so that the lanes will be clear lines to later be detected using Hough Transform. The first simple approach was to convert the image to grayscale and then binary (based on intensity), clean road noise using a median filter, and use canny to find the lane edges. Additionally, I first started with a trapezoidal ROI mask that would constrain the area where the algorithm looks. The trapezoidal ROI mask wasn't robust enough because the road was too noisy and it didn't handle shadows well – lanes in shadows disappeared and it would have adapted poorly to night view. The preprocessing was made more robust in a second attempts, having the following steps:

1) Convert the frame to the HLS color space - similarly to HSV it is easy to extract just the brightness values of pixels, with the difference being that with HLS I can just take the lightness value without caring for the hue and saturation (good for

detecting yellow lines as well – and any bright color for that matter). Since lane lines are white and the road is a darker grey, the heuristic was that the lane will always be brighter than its surrounding road.

2) Clean road noise using a median filter
3) Use a “Top-Hat” filter to remove differences in brightness – this technique involves heavily blurring the image and subtracting those low frequencies from the unblurred image. This leaves only the pixels that are locally brighter than their neighbours, and effectively leaves us with only locally bright edges.
4) The previous image was good but still left road noise and cracks in the road. The next steps were to use adaptive thresholding (threshold based on the local average of a neighbourhood’s brightness) to convert the image to binary, and then clean it up a bit using another median filter and erosions and dilations.
5) The result from the above was getting better, but there was still a non-negligible amount of road noise, however, almost entirely at the bottom of the image (where the road is more detailed). To tackle this, I built a custom geometric filter based on blobs – identify all blobs (connected components) in the mask, identify their y coordinates, threshold based on their size and y coordinate. This let me keep smaller blobs further up the screen while deleting small noise at the bottom (small noise at the bottom could be the same size as a far lane).
6) For the edge detection, I applied canny to both the (slightly blurred) lightness channel image and the filtered blob mask and did a bitwise and between them. This got rid of all the noisy edges from the original image, and also got rid of edges of dark cracks in the blob mask that survived the filtering.
7) Finally, I applied the ROI mask. This led to the following result:



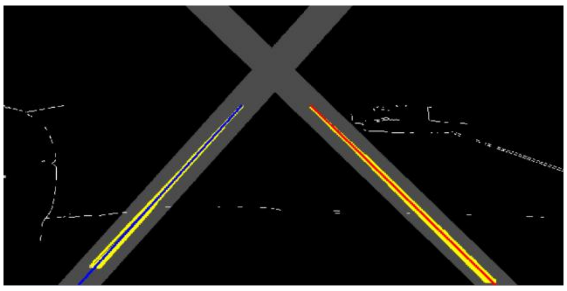
The green trapezoid is the ROI and the pink one is the middle of the lane which we ignore. As you can see, we have minimal road noise and we clearly see the lanes.

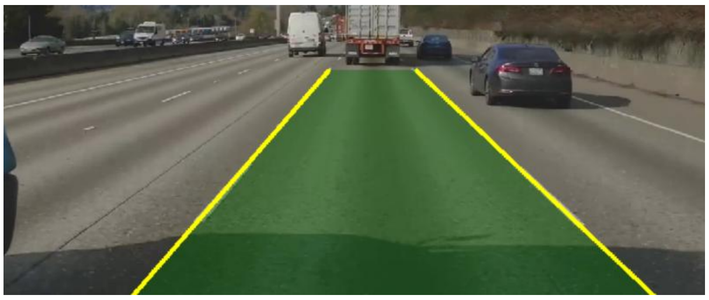
I then moved on to tracking the lanes. To do this I applied the following actions:

1) Apply Hough Transform to the pre-processed mask and convert the lines to polar coordinates so that it will support vertical lines.

2) Filter out noisy lines that are too horizontal and average the detected lines as the new lane line. Also added an anchor point where the previous line was to prevent extra movement.
3) Average the new lane line with the last few lane lines (history) to make sure it is smoother and doesn't change too much.

This method yielded good tracking of the lanes. However, since the ROI was fixed during lane changes the lanes would get completely lost as the ROI wouldn't cover them properly. For this reason, the ROI was used only to detect the lanes for the first time (when there wasn't a lane already detected). Once I had a lane, I would only consider lines within a sleeve around that lane, with the sleeve expanding in case there was a lot of movement to allow for sharper changes. This is what it looks like:

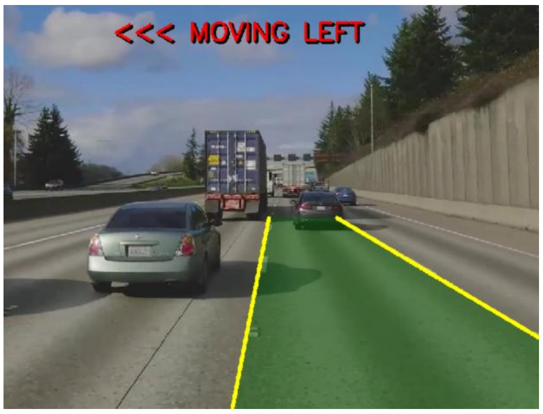




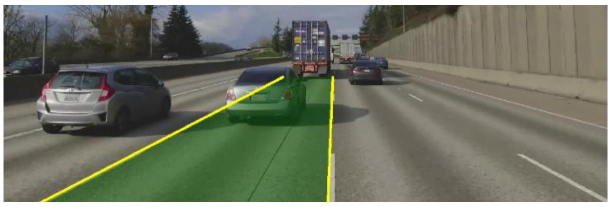
This made it so that the detected lanes would stay on the real lanes until told otherwise.

Finally, I had to add support to lane changes. At this point I had a key observation – when the lane changes, the lane on the side of the change becomes vertical (or almost vertical) at some point. This was the basis of the lane change logic – when the lane becomes close to vertical then we signal that a lane change is happening, and additionally, we can reflect the other lane with respect to the vertical lane to get very close the lane one the other side, visually this means:

When the left lane becomes close to vertical, I can warn that a lane change is happening to the left:



Once the lane was vertical enough, I reflected the right lane with respect to the left lane, immediately landing on the next left lane. This let the lane detection continue smoothly.



This worked very well but there was one small optimization I wanted to tackle with lane tracking – since the lanes were free to move they could sometimes move too far away from each other if there were noisy vertical lines such as long white truck stripes. Additionally, if one side of the lanes was too faint to detect then one of them could get lost while the other was firm. For this I used an estimate width of the lanes – I didn't let them get too far away from each other and kept them at some distance range from each other.

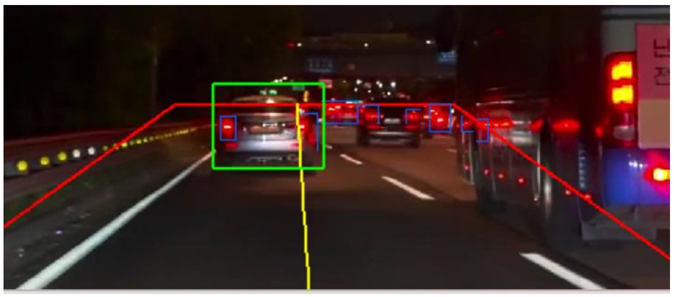
<u>**Part 2 – Adjusting for the Night:**</u>

Frankly, since the preprocessor was already designed to handle different lighting conditions, this part was free other than tuning some parameters to fit the new video better such as adjusting the initial search zones for the new camera setup, lighting thresholds, and lane change angle thresholds (this was affected by the camera position and the "flatter" road in the video). A peculiar parameter change made for this video was to use a history of only a single lane. This was chosen because using more than that limited the amount of change the lane could do between frames (because it gave weight to old irrelevant frames), and in this video the lane changes happened much faster than the previous one.

<u>**Part 3 – Car Proximity Detection:**</u>

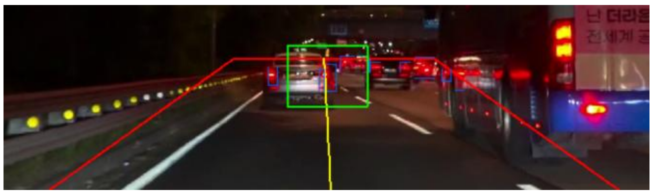
In this part I had to detect the car directly in front of me and approximate its distance from me. To detect the car, I attempted to detect its taillights which were bright red since the video was in the dark. This involved making a red mask using an appropriate color space, defining the ROI, and selecting the correct taillights.

At first, I used a static ROI and just searched for the two most centre taillights. However, this proved to be not good enough – many times (especially when the road is not perfectly straight or when the car in front of me wasn't perfectly centred on the road) the middle lights would miss and go on to a different lane or far away cars. To combat this, I used my detected lanes to point towards the centre of the lane and check the two closest taillights from each side. This looked like this:

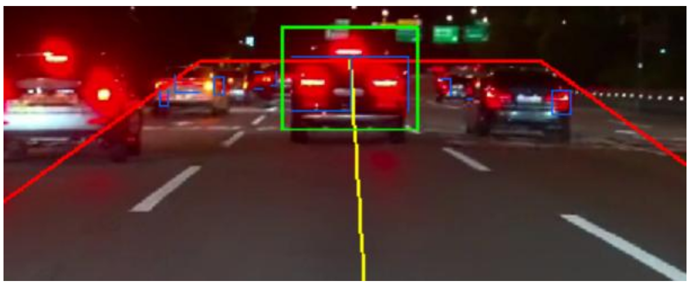


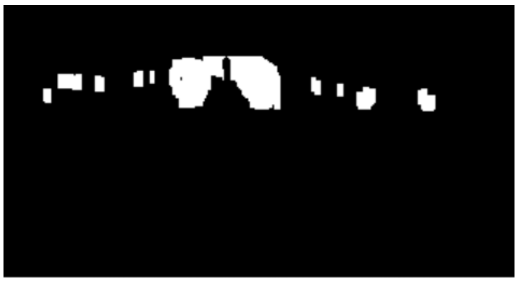
The red trapezoid is the ROI, the yellow line is a mix of the centre of the screen and the centre of the lanes, and the box is the predicted detection.

However, this was inaccurate often because the car in front was not aligned with the lane well, which led to results such as:



Sometimes there were taillights that were connected throughout the entire back of the car. To combat this, I forced a split through the centre line to make sure that those were the detected taillights:





On the right is the red mask, without forcing the the black line where the center line passes, it would have chosen the big centre blob and one of the farther away ones, failing to achieve a good bounding box on the car.

The bounding box scales depending on the y coordinate of the taillights – closer taillights had larger boxes and farther away ones became smaller.

To estimate the distance I used the pinhole camer model:

$$Distance = \frac{FocalLength}{y_{screen} - y_{vanishingPoint}}$$

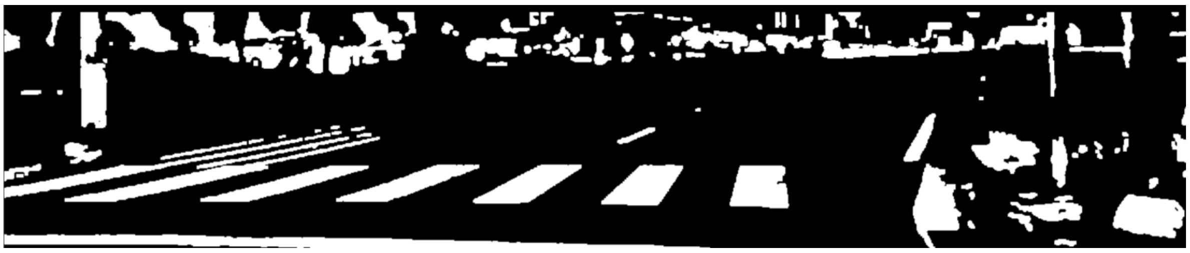
I tried calculating the focal length based on the length of a lane stripe in the real world and on camera, but it ended up that guessing a focal length gave me numbers that seemed more true to reality to my eye. The vanishing point was dynamically calculated based on the detected lanes in each frame. The color of the boxes was determined by the estimated distance of the car – green is far and safe, yellow is moderately close, and red is dangerously close.

This part had the most consistency problems, especially because this method is vulnerable to irregular movements and is reliant on consistent behaviours. If I had the time to redo this, I would maybe try to use the daylight video and turn cars into blobs of colors and track them.

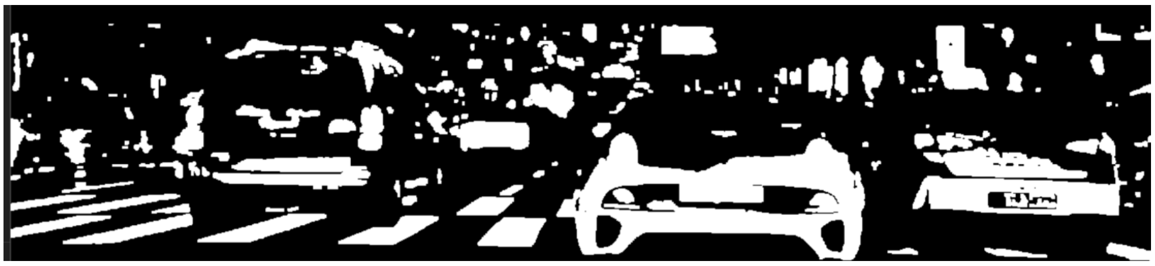
<u>Part 4 – Crosswalk Detection:</u>

The initial approach to detect crosswalks was to use a barcode scanner method – I pre-processed the image to keep bright pixels (since the crosswalks are white), split the bottom part of the image into several horizontal lines and treated each one as a signal. I then looked for frames with consistent signals of the form "black-white-black-white…", and if I had enough of them which were similar (the variance of the width was low), I marked their area as a crosswalk.

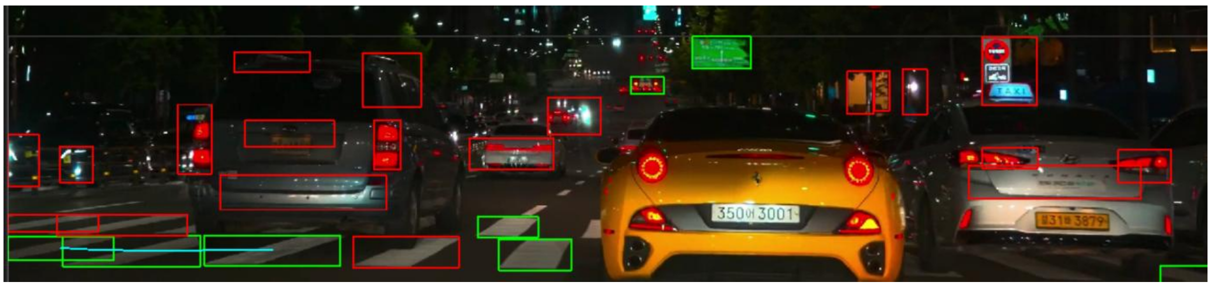
I quickly realized that this simple preprocessing didn't detect the crosswalks well enough because of the varying lighting conditions in the video, so instead of it I used the pre-processor from the lane detection which was already good with handling different lightings. I did make a slight tweak to it – I added a flag that, when activated, made it return the calculated blobs rather than the edge mask. Additionally, the relative brightness kernel I used was larger than the crosswalk bars so it would keep them in tact and not hollow them out. If I kept the smaller kernel, then the insides of the crosswalk would not appear brighter than their surroundings (they are all white) and they would have been filtered out. The pre-processed crosswalk mask looks like this:



The barcode method was good for a proof of concept but failed in later parts where the crosswalk was occluded or was irregular such as:

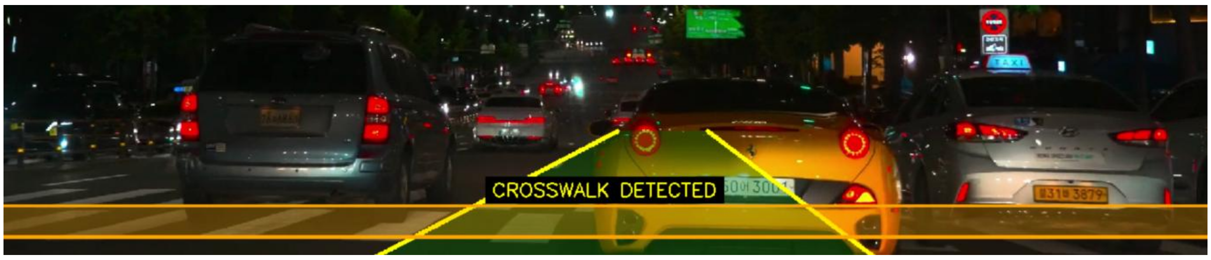


Because of this, I switched to a different method – looking for rectangular(ish) blobs that are a relatively similar distance from each other. If I had enough of them, I connected them:



This is what that part looks like. All the boxes are rectangularish blobs detected by the mask above, some of them are too small or have the wrong aspect ratio (which was tuned to represent crosswalks) and therefore got filtered out, and I marked crosswalks only if three boxes in a row had similar distance and were consistent enough (the boxes with the blue line between them).

As you can see, this doesn't detect the entire crosswalk all the time and is sensitive to slight lighting and angle changes. To get around this, I used the fact that crosswalks change predictably with the car movement and that I didn't have any false positives. This let me just draw the box of a rectangle on the entire width of the screen and track it until it goes down out of the screen according to the camera movement:



<u>Additional Notes:</u>

In the last video I understood a major limitation that my lane detection algorithm had – it could not track sharp turns or congested lanes well at all. It completely lost track of the lanes, to the point that I had to add a reset feature for this exact case (if it doesn't see a lane for too long then it resets itself as if it is the start of the video). There is a major part of the video where it fails to find a lane after it loses it because of a turn and then because of irregular lanes and cars that are too close. Once the road does clear up a bit it does manage to find the lane again and keep tracking it even with close cars in front of it (because the lane is still visible at that point). I couldn't find a better video for the crosswalks that had all the requirements and a clear enough road, so I chose to accept this flaw.

The unsung file of this report is the process_video.py file. This is the orchestrator that handled all the different debug views (the screenshots attached throughout the report) and kept all the settings and parameters for the videos. This was essential for live debugging (in 10fps...) and parameter tuning.

Final Discussion:

The main strengths of my project, are the robust pre-processor and the lane detector – especially the lane switching logic (the reflection). A way to make it even more robust is to have the lanes keep an angular relation – their slopes have a relationship that can be tracked and enforced to make sure they are more consistent with each other. Its weakness, as previously explained, is that it doesn't track sharp turns very well (although it handles softer curves alright). Overall, I am happy with the lane detection and robust, lighting invariant, pre-processor.

The proximity detection is the weak point of the project, and can potentially be improved using the suggestions I wrote above. This task is notably hard.

The crosswalk detection works fairly well, with the potential improvements being detection from farther away and handling the crosswalk at an angle.

This project was a significant learning experience, that demanded a lot. I hope the results are satisfactory and that the report demonstrated the immense amount of effort put into this.

VIDEO LINKS:

LANE DETECTION HIGHWAY DAYLIGHT:
https://www.youtube.com/watch?v=X6ND38tKoSM

NIGHTTIME PROXIMITY:

https://www.youtube.com/watch?v=o2C_oNhlK5U

NIGHTTIME CROSSWALK:

https://www.youtube.com/watch?v=GSAxxWnDdlc