Analysis, Explanation and for Lane-Detection

Raúl López Musito
R3FOX1
ELTE
IFROS 4

Github link: https://github.com/Raulmusito/LKA

1. Summary

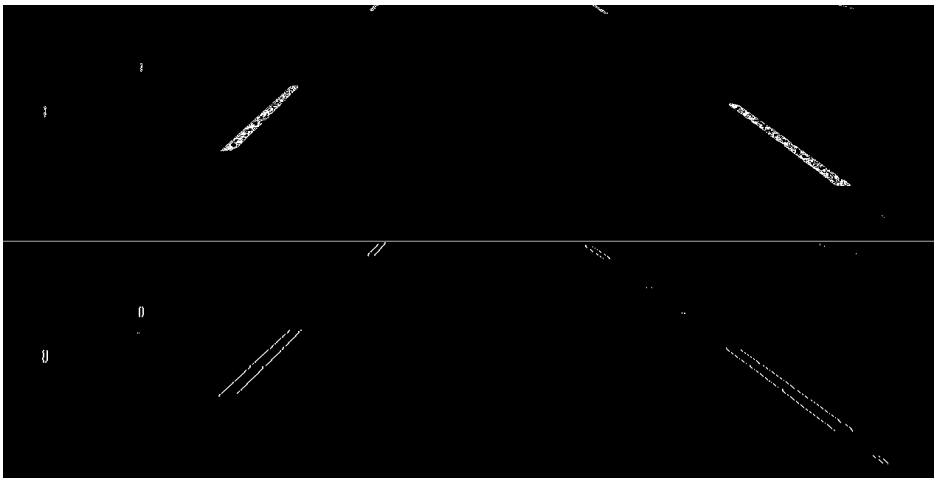
This Python script is an image-processing pipeline that detects lane lines in a sequence of consecutive frames, extracts lane candidates via color and gradient masks, transforms the image to a bird's-eye view (IPM), finds strong vertical (column) peaks in the IPM image histogram, fits straight lines to those peaks to estimate lane geometry, and overlays the estimated lane region back onto the original frames. The code mixes OpenCV image processing, NumPy arithmetic and a little SciPy peak detection.

---

2. High-level pipeline

1. **Preprocessing**

    - Each frame is cropped to the bottom half (drivable region).
    - Converted to HLS color space and split into channels.
    - A saturation mask (`s_mask`) highlights colored parts (S channel), and an intensity threshold on L combined with a Sobel X operator (`sobel_mask`) detects bright vertical edges. In this stage the thresholds where set to be very

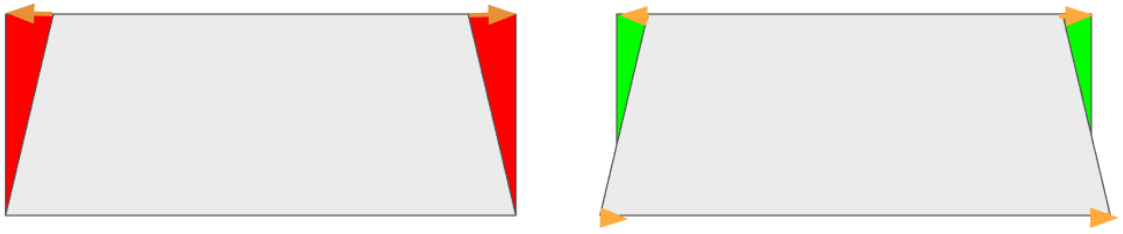


strict, keeping as much noise as possible out of the system. This can be seen in the image (upper frame is the s_mask and lower frame is the sobel mask) that this was actually accomplished.

- The S and Sobel masks are combined (bitwise OR) to get candidate lane pixels.

2. **Inverse Perspective Mapping (IPM)**

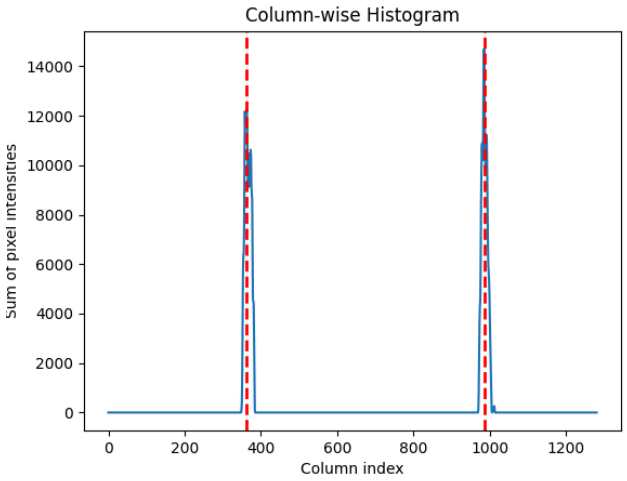
- Hardcoded src and dst corner points are used to compute a perspective transform matrix **M**, which converts the image from the car's point of view to a bird's-eye (top-down) view. The combined mask is then warped to Inverse Perspective Mapping (IPM) coordinates.
- In the improved configuration, instead of moving only the upper corners outward to simulate the bird's-eye perspective, all four corners were adjusted toward the image's midpoint. This modification reduces the amount of perspective distortion and scaling, especially near the upper edge of the frame. Resulting in a more uniform and stable top-down projection.



- It can be observed in the image that portions of the original frame near the left and right edges are lost due to the scaling introduced by the perspective transformation. However, this does not affect the algorithm's performance, since the lane markings of interest are typically located near the center of the image.

3. **Histogram & Peak Detection**

   - Column-wise sum of the IPM image produces a histogram of likely vertical lane positions.
   - The two main peaks of the histogram are selected as lane candidates. In addition, a proximity filter is applied so that two peaks cannot be closer than 200 pixels to each other. This ensures that noise or minor fluctuations in the histogram are not mistakenly identified as separate lanes but are instead treated as part of the same lane. The 200-pixel distance threshold was manually tuned based on empirical observations.



4. **Line fitting in IPM**

   - For each peak, `fit_lane_line_from_peak` collects per-row x-positions and fits a polynomial (degree 1 by default, i.e. linear) x = m*y + b. This yields an estimate of lane position as a function of image row (y). This step also returns:

'm' : slope (dx/dy) of fitted line (float)

'b' : intercept (x = m*y + b)

'angle_deg' : angle in degrees relative to vertical (positive -> leans right as y increases)

'x0,y0' : line endpoint at y=0 (float x0 may be non-integer)

'x1,y1' : line endpoint at y=h-1

'points' : (y_coords, x_coords) arrays used in fit

- The xn ,yn are already indicating the position of the detected lane. However, they are still in the IPM pov.

5. **Back-projection & display**

   - Line endpoints are converted back to original image coordinates via the inverse homography **`invM`** and drawn on the cropped frame; the lane polygon between left and right lines is filled and blended for visualization.
   - To plot the egoline (blue polygon) a trapezoid is done with the same 4 points of the lanes.

6. **Compute error per lane.**
   - A text box located in the upper-left corner of the image displays the fitting error for each detected lane, expressed as the Mean Absolute Error (MAE). This calculation is performed while the image is in theIPM view. The error is obtained by comparing the actual lane pixel positions—computed as the mean x-coordinate of detected lane points for each image row—with the corresponding x-values predicted by the fitted lane model. In this way, the MAE quantifies how accurately the fitted line represents the observed lane structure in the IPM perspective.