SE498 Introduction to Autonomous Vehicle System
Laboratory Assignment 4.2: Lane detection

Goals for this Lab Assignment:

1. Learn to use PerspectiveTransform to convert image to top-down view for later processing.
2. Use Region of Interest (RoI) method to find points on lines that define lanes.
3. Use least square method to fit polynomial to the points that define each line.

Setup:

1. Download lanefollow.tar.gz and usb_cam.tar.gz on PC (or raspicam_node.tar.gz on raspPI).
2. Extract to your workspace.
3. You will be mainly modifying img_processor.cpp in lanefollow/lane_follow/src/ folder. You can modify main.cpp and /lane_follow/include/lane_follow/img_processor.hpp header file if needed.
4. To run the driver for the usb camera, use: roslaunch usb_cam usb_cam-test.launch
5. To run the lane follow code, use: rosrun lane_follow run_lane_follow

Understanding the basics:

1. The imgProcessor class declared in main function will automatically subscribe to the camera topic defined in its constructor. Please modify accordingly depend on which platform you are working on (PC or Pi).
2. The subscriber callback function will always receive the latest pointer to the image being published and store the pointer inside the class as a private variable, until a new image is being publish to the topic subscribed. A new pointer to the image will effectively overwrite the current pointer and cause the old image to be destructed (smart pointer self-delete).
3. Inside the main function, a while loop will call getLastMat(output) every 1/9 of a second, and grab the current stored image, and store it in "output".
4. Then the image will be resized to cv::Size(410,308) to reduce computation load on pi.
5. The resized image will be put into function imgProcessor::roadDetect(output) for further road detection processes.
6. The perspectiveTransform matrices are also defined inside constructor of the class. Modify accordingly.

Main Objective: Finish the code in roadDetect().

NOTE: ctrl+f "@todo" to find what you need to do inside the code

There are few steps to a rudimentary road detection algorithm.

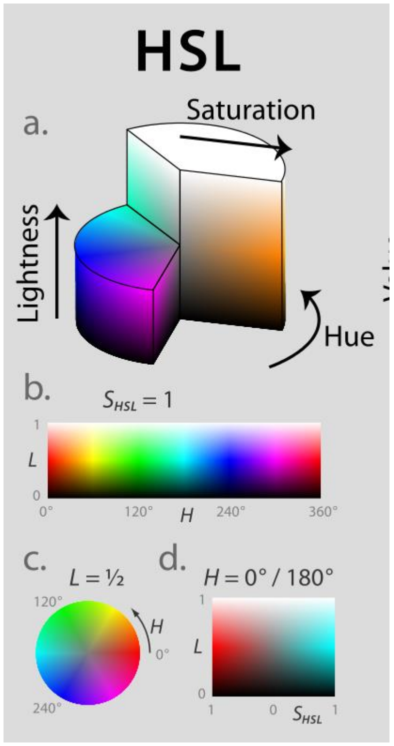
1. Do a perspective transform on the current image to a bird eye view (top-down) for a more straightforward image to process.
2. Convert the RGB color to HLS so that lighting conditions will have less of an effect on our algorithm.
3. Do edge detection on the image we have, to find distinct edges of the marking on the ground (or the edges of the line).
4. Uses Region of Interest (RoI) method to find the points on the lines, more on this later.
5. Curve fitting on the points that represents each line.

Part I Perspective Transform

To simplify the work, we can transform image we have to an image of the road from the perspective of a bird in the sky. To extract all the information we need to warp this image from a vehicle view to sky view, we just need location coordinates. Specifically, all we need is one image which we have some locations from the input perspective (vehicle view) and the respective locations of the desired perspective (sky view).

1. Assume that your camera will be looking at the groud at a -45 degree angle. Modify the function getPerspectiveTransform(), specifically, the locations from the input perspective and location of the desired perspective.
2. You may need to fine tune and modify it later on the cart.
3. The transform should be able to convert image from the -45 degree angle setup to a top-down view.

Part II Edge Detection



First thing we do before edge detection is to do a conversion from RGB to HSL. The Hue value is its perceived color number representation based on combinations of red, green and blue. The Saturation value is the measure of how colorful or how dull it is. Lightness is how closer to white the color is. This has been done for you in the code by simply calling opencv function. The reason for such conversion is that we are only interested in the color of the lanes, not the lightness nor the saturation. It makes our algorithm more robust to lighting changes. See code for detail.

After the conversion, we can start the edge detection. In this case, we are using a Sobel operator. A Sobel operator essentially measures the rate of change in value (and its direction) between two locations of the image. This is also technically called as the derivative or gradient. We can define in which direction of the image we want to have the gradient calculated. Since the lanes are more likely to be in a vertical orientation than horizontal, we want to take Sobel operator in HORIZONTAL direction.

1. Take Sobel operator in HORIZONTAL direction
2. Normalization step should be taken to normalize the sobelx to 0-255 range

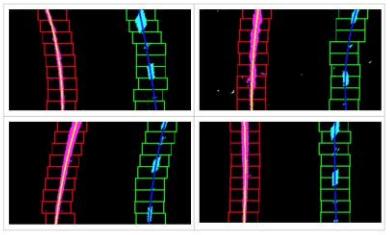
See @todo in the code to find out what you need to do. Once you have implemented the code, you should be able to see vertical edges to be more prominent.

Part III Color Selection

Now assume we know what color we use for the lane, we can select what color needed to be filtered for optimal road detection.

1. Inside select color, fill in the cases for each type color to be filtered based on HSL.

Part IV Find Curve

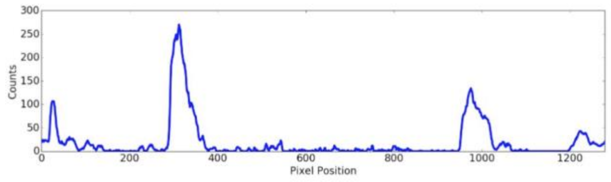


Given the detected edges and the color filter, we can do an AND operation on both mask to generate a combined mask that indicate edges of specific color.

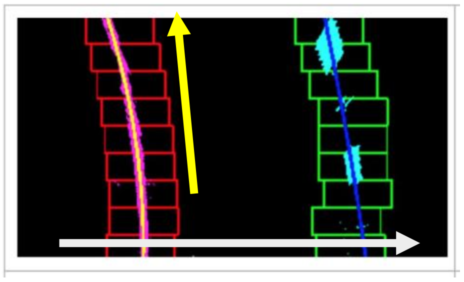
Now I do a "sliding RoI"—one RoI on top of the other that follows the lanes up the frame. The center of a "region" are added to the list of points in the lane. I can take the histogram of all the columns of the RoI and I will get graph with one peak. The prominent peak of the histogram is a good indicators of the x position of a lane. This gives a good indication of the base point of the next window above. We can repeat this over and over until we get on top the lane. This is implemented using recursion.

This way we have accumulated all the points that we are interested in that we will feed to our polyFit() function which spits out the coefficients of the 2nd degree polynomial that best fits the points for each line.

In this part, you will need to



1. Finish the getHist() function that will return a histogram x-axis as x position of the pixel, and y-axis as sum of intensity of the column of pixels in x position.
2. Finish the polyFit function to find the coefficients of a polynomial function that best fit a set of data points. Specifically, find a, b and c such that x = ay^2 + by + c where x and y are the coordinates of the points in the lane. The solver to solve a least square fit problem in the form of Ax=b has already been provided, you will just need to design A and b correctly.
3. Fill in the findCurveRecur() function's recursive part, such that the behavior of the findCurveRecur() is as such:
    a. Using RoI, recursively traverse the bottom row region only, return when recursively hit the right most column. See white arrow.
    b. If a lane is found, recursive travel upward only, following the lanes until the top row is reached, then return. See yellow arrow.
    c. Store the points found along each line to the correct line storage vector.
    d. You can draw boxes to help you visualized your recursive function and to debug.



Once all have been implemented and tested, you should be able to see curves that should follow the lanes.