Project 4 – Advanced Lane Line Detection
Written by: Graham Nekut
Date: April 15th, 2017

Files

- P4_advlanelines.py: Image/video pipeline for lane line detection.
- Project_video_out.mp4: Video recording with projected lane lines, turning radius, and center offset.
- Images/image_threshold.png: Images of all gradient and color channel thresholds.
- Images/lane_detection.png: Images of full image transformation and lane line detections (distort, combined thresholds, perspective, line detection, perspective line projection)

Process

- **Camera Calibration (Line 14-42, 420):** All cameras inherently distort real work images due to the nature of projecting 3-D objects into a 2-D plane and lensing effects (light transmission slightly off at the edges of lens, lens tilt, etc.). Camera calibration helps remove those distortions by calculating transformations from a set of images that have a known output structure. To calibrate, a set of checkerboard images were fed into OpenCV functions (findChessboardCorners, calibrateCamera), returning the calibration matrix and distortion coefficients. These parameters can then be used for all images from the camera to correct for distortion.

- **Distortion Correction (Line 315):** Using the calibration matrix and distortion coefficients from the camera calibration step, image distortion is corrected for with the OpenCV undistort() function.

- **Gradient and Color Threshold (Line 73-139, 317-323):** Different threshold and color selection techniques were used to better visualize key features within an image. Using the Sobel operator (a simplified method for performing weighted sums across an array of pixels), each image was translated into directional x/y gradients, which were combined to find the magnitude and direction of the gradient. The RGB/BGR images were also translated to a different color spaces to reveal other attributes of the image. Distinct colors differences on the road (such as yellow or white lines on a black surface) were made clear by selecting the saturation channel (S) within the Hue-Lightness-saturation (HLS) color space and filtering to those pixels within a certain range. All of the gradient and channel images were converted to binary pixel mappings by setting all pixels that met the defined thresholds to one and those that did not to zero. Simple and/or logic were then used to combine the various images into a singular view that met the threshold criteria. ***See Figure 1.***

- **Perspective Transformation (Line 45-71, 325, 337):** Images were converted to a top-down view (i.e., birds-eye) using OpenCV functions (getPerspectiveTransform, warpPerspective) to help better detect lane lines and fit a polynomial curve. The functions take two four-point perspective mappings (source image, destination image) and determine the proper matrix transform to map points in the image to a new image. To verify that our transform source point were set correctly, a straight-lane highway image was used. If set properly, the transformed top-down image will have parallel lane lines. Once the perspective is calibrated properly it does not have to be adjusted, so long as the car camera is in the same orientation to the horizon (i.e., no hills). The inverse perspective is also necessary to map the detected lane lines in the top-down view to the actual camera perspective; this matrix was calculated by switching the source and destination mapping points. ***See Figure 2.***

- **Line Detection (Line 161-298, 327):** Determining the polynomial best-fit line from a projected, threshold image of lane lines. Initial line determination uses a moving window approach, starting at a base point (xlb_/xrb_) where the concentration of valid pixels is highest in the bottom quarter of the image. A list is maintained that contains all pixels that lie within a predefined margin (x_marg) to either side of the base point within each y-window. If enough pixels are found within the window, a new center is calculated that will serve as the base for the next window. This process continues for the total number of input windows (win_cnt, greater number of windows, better line resolution but caution of noise and performance). Once the image is fully scanned, a second-degree polynomial is fit to all of the valid pixels; this fit line is the images approximation of the detected lane line. The computed polynomial coefficients are saved in an object array and the process will move on to the next image frame. Since lane lines do not change drastically from image to image (assuming the frames per second are high enough and the driver is not driving erratically) the previously computed N polynomial coefficients are utilized to find the next images lane pixels by searching within margin (x_marg) of the calculated polynomial. Using the lane line history can greatly help to smooth out lane detection from frame to frame. ***See Figure 2.***

- **Center Offset Calculation (Line 264, 275):** Assuming that the camera is mounted in the cars center, the center of the image should also correspond to the center of the road. Once knowing the polynomial fit, the x-coordinates predicated by each line should average out to the images center. Any deviation from that will be equivalent to the number of *pixels* the car is

off of center. To convert to actual distance, multiply by the ratio of actual lane width to pixel lane width (e.g., 3.7 meters per lane/700 pixels per lane)

- **Radius Calculation (Line 181-194, 273-274):** The radius was found by fitting a polynomial to the actual image by scaling the found pixel coordinates from the top-down view to real distances. Similar to converting the center offset distance, each x- and y- coordinate are converted to real distances by a conversion (x: lane width, y: distance to a fixed point). Once the new polynomial is fit to the new collection of coordinates, an equation can be derived to return the radius of a given polynomial function at a point (the point chosen was the base of the image).

- **Visualization and Video Processing (Line 342-378, 422-424):** Visualizations are provided in Figure 1 and Figure 2 (shown below, commented out in video pipeline). To process the video, code from project 1 was recycled and modified to fit current needs. For projection of turning radius and center offset, OpenCV provides a putText() function that allows for any text to be added on top of an image.

Discussion

Each part of the project could easily be broken apart into modules and I was able to develop them independently. One of the hardest things to get right was what threshold and threshold images I should use/combine to get the best resolution of lane lines under changing conditions. Once I had some values that work fairly well, I moved on to trying to detect the lines that I had filtered for. Luckily, my parameter selection was reasonable enough to provide a nicely smoothed out lane detection (once I completed the pipeline and polynomial smoothing).

A few ways that I could improve upon my existing pipeline:

- Threshold selection: Changing the actual threshold values used for binary filtering, Sobel kernel size for smoother gradients, what combinations of threshold images to use, adaptive selection of threshold images depending on lighting conditions (switch from saturation (S) to red (R) under different conditions),
- Polynomial vs. window-search switching: This was definitely something that I would need to get the challenge problems to work properly. My current design relies entirely on the history of the polynomial coefficients found; however, that does not work when the lane deviates from the previous behavior. Determining what can be classified as a deviation is the hardest part (lane line polynomials aren't parallel, predicted radius is outside of the norm, number of pixels detected is not reliable, etc.). In any deviation there are two ways to handle it: 1) if previous measurements were good and lane lines cannot be detected in the current frame, keep going on that same path (i.e., use the polynomial), 2) if previous frames were relying on an older polynomial and a new window search finds a new valid polynomial fit, clear the previous history and start storing the new parameters.
- Performance in other weather conditions: Curious how to adapt models based on snow, sun, night, etc.

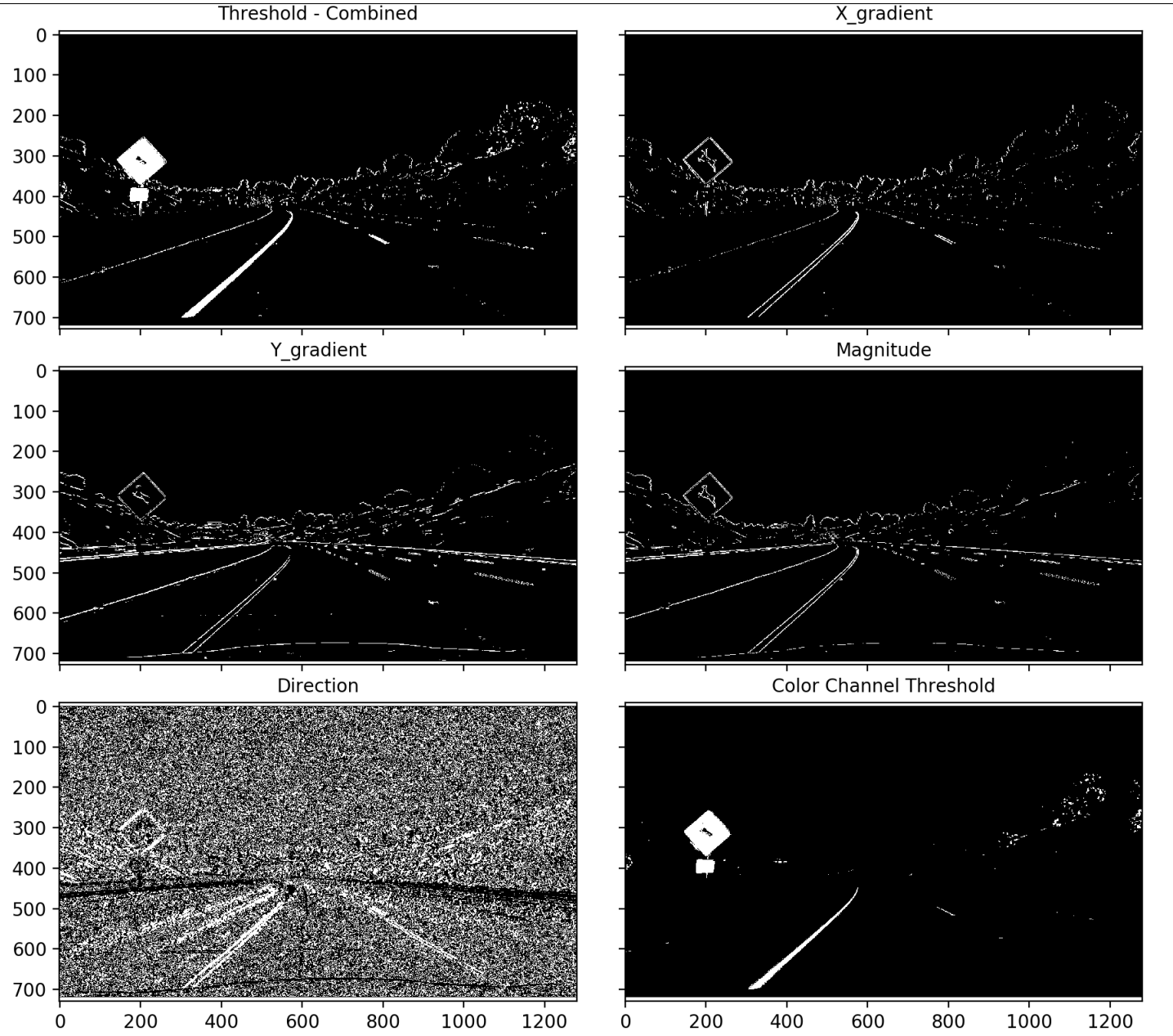

Figure 1. Image grayscale gradient and color channel threshold detection. From top left, read left to right: 1) Combined image thresholds for clear lane line detection (uses saturation color channel, x-direction gradient, and magnitude/direction gradient combined with or-logic), 2) x-direction gradient with Sobel kernel of 3, 3) y-direction gradient with Sobel kernel of 3, 4) Magnitude gradient (i.e. high overall contrast), 5) directional gradient filtered to horizontal gradients (i.e., vertical lines), 6) saturation color channel after BGR to HLS conversion.

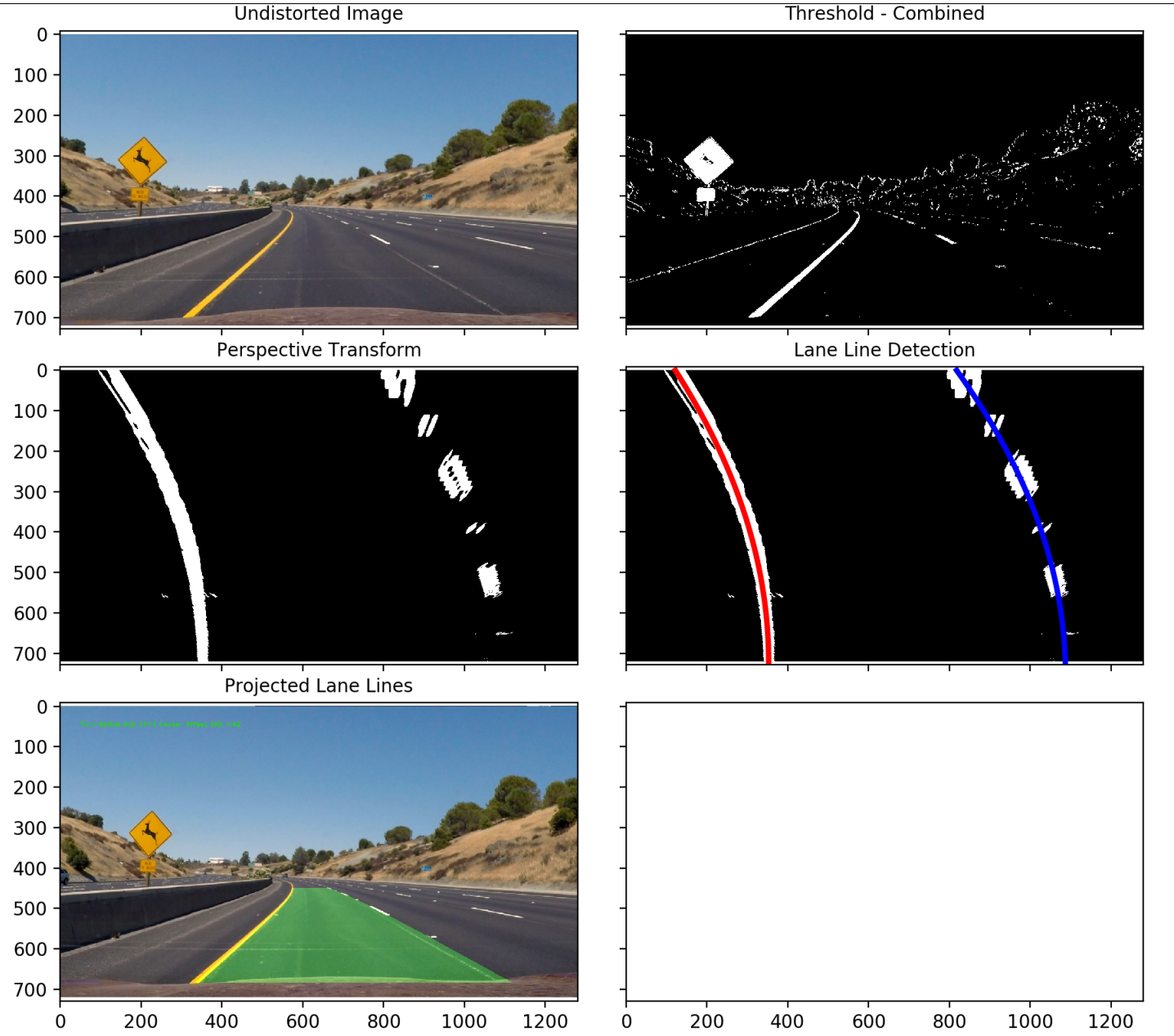

Figure 2. Full image pipeline. From top left, read left to right: 1) Undistorted image using the calibration matrix and distortion coefficients from camera calibration, 2) Combined gradient and color channel threshold images, 3) Perspective transform of threshold image, 4) Polynomial fit from lane line detection, 5) Lane lines perspective projection.