

Hoàn thành

Mục tiêu công việc: Xác định nhận dạng làn đường trong một đoạn video từ một camera phía trước trên một chiếc xe hơi. Phát hiện các tuyến đường trong nhiều điều kiện, bao gồm thay đổi đường bộ, đường cong, và ánh sáng thay đổi.

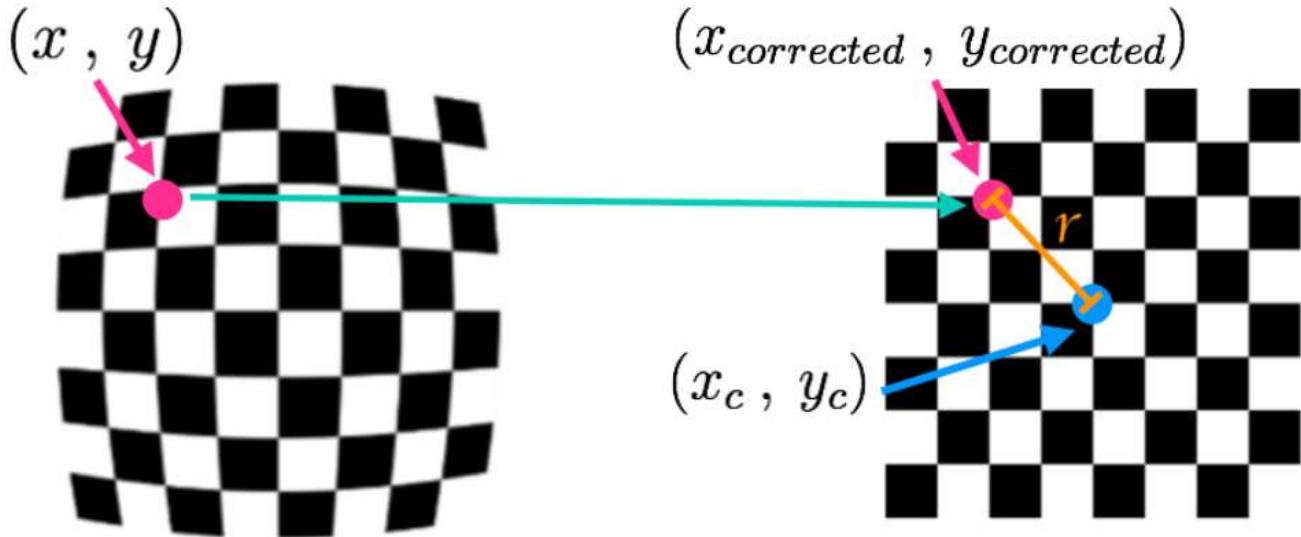
+ Hiện tại xử lý tốt môi trường lý tưởng là ban ngày có nắng mạnh hoặc vừa và góc quay camera không đổi

Các bước thực hiện:

1. Tính toán ma trận hiệu chuẩn camera và hệ số biến dạng cho một tập hợp các hình ảnh bàn cờ vua.
2. Chỉnh sửa méo từ ảnh thô.
3. Sử dụng các phép biến đổi màu và gradient để tạo ra một hình ảnh nhị phân để phát hiện làn đường. (ở bước này ngoài cách trên ta còn có thể dựa vào ngưỡng màu để tìm ra màu ta muốn tìm trong các không gian RGB, HSV, HSL, YCrCb, Lab)
4. Áp dụng một phối cảnh chuyển đổi để khắc phục hình ảnh nhị phân ("birds-eye view").
5. Phát hiện điểm ảnh làn và phù hợp với một biểu thức đa thức để tìm ranh giới làn đường.
6. Xác định độ cong của làn đường và vị trí của xe đối với trung tâm.

Bước 1: Giai đoạn hiệu chỉnh Camera

Camera thực tế sử dụng các ống kính cong để hình thành một hình ảnh, và các tia sáng thường uốn cong quá nhiều hoặc quá ít ở các cạnh của các ống kính này. Điều này tạo ra hiệu ứng bóp méo các cạnh của hình ảnh, do đó các đường kẻ hoặc các đối tượng xuất hiện nhiều hoặc ít cong hơn thực tế. Đây được gọi là biến dạng xuyên tâm, là loại biến dạng phổ biến nhất. Có ba hệ số cần thiết để biến dạng xuyên tâm: k_1 , k_2 , và k_3 . Để sửa sự xuất hiện của các điểm bị bóp méo hình ảnh trong một hình ảnh, người ta có thể sử dụng một công thức chỉnh sửa được đề cập dưới đây.



$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Trong các phương trình sau, (x, y) là một điểm trong hình ảnh méo. Để undistort những điểm này, OpenCV tính r , đó là khoảng cách được biết giữa một điểm trong một hình ảnh undistorted (đã chỉnh lại) ($x_{corrected}$, $y_{corrected}$) và tâm của ảnh bị méo, mà thường là tâm của hình ảnh đó (x_c, y_c). Điểm trung tâm này (x_c, y_c) đôi khi được gọi là điểm tâm bóp méo. Những điểm này được mô tả ở trên.

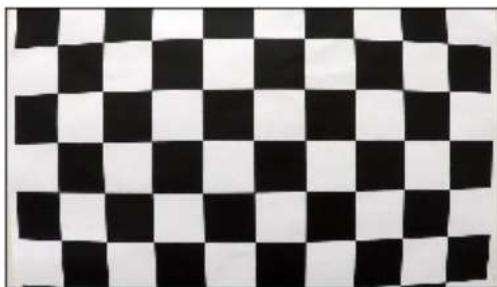
Chức năng do_calibration () thực hiện các thao tác sau:

1. Đọc ảnh chessboard và chuyển đổi sang ảnh xám gray scale
2. Tìm góc của chessboard .
3. Bắt đầu bằng cách chuẩn bị các điểm đối tượng, nó sẽ là tọa độ (x, y, z) của các góc ảnh bàn cờ trên thế giới. Ở đây tôi giả định rằng ảnh bàn cờ đã được cố định trên mặt phẳng (x, y) tại z = 0 sao cho các điểm đối tượng giống nhau cho mỗi ảnh hiệu chuẩn. Do đó, objp chỉ là một mảng được lặp đi lặp lại của các tọa độ, và objpoints sẽ được nối với một bản sao của nó mỗi khi tôi phát hiện thành công tất cả các góc chessboard trong một hình ảnh thử nghiệm. imgpoints sẽ được nối với vị trí pixel (x, y) của mỗi góc của mặt phẳng hình ảnh với mỗi lần phát hiện ảnh bàn cờ thành công.
4. Thực hiện cv2.calibrateCamera () để tính hệ số co-efficients và ma trận camera mà chúng ta cần phải chuyển đổi các điểm đối tượng 3d thành 2d điểm ảnh.
5. Lưu giá trị hiệu chuẩn tệp camera_cal.p để sử dụng sau.

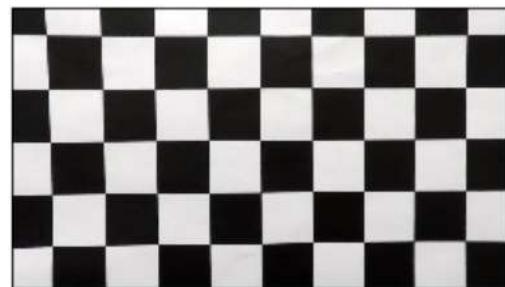
Bước 2: Sửa méo ảnh

Sử dụng các hệ số biến dạng và ma trận camera thu được từ giai đoạn hiệu chỉnh camera, để loại méo ảnh sử dụng chức năng cv2.undistort.

Một ảnh cờ vua mẫu và hình ảnh không bị bóp méo tương ứng :



Original image



Distortion corrected image

Bằng cách hiệu chỉnh méo, chúng ta thấy rằng các đường chessboard xuất hiện song song với hình ảnh thật ban đầu

Một hình ảnh mẫu khác và hình ảnh sửa méo tương ứng được hiển thị bên dưới:



Original image



Distortion corrected image

Có thể thấy rằng chiếc xe bên phải xuất hiện sẽ được shift đi so với ảnh gốc ban đầu.

Bước 3: Tạo một ảnh nhị phân

Trong giai đoạn nhị phân ảnh, nhiều phép biến đổi được áp dụng và kết hợp lại để có được hình ảnh nhị phân tốt nhất để phát hiện làn đường.

Để chi tiết hóa bước này ta lấy 8 ảnh sau làm ảnh đầu vào:



3.1 Dùng ngưỡng trong các không gian màu

Đơn giản ta tìm ngưỡng để lấy ra làn đường màu vàng và màu trắng

RGB

In []:

```
# white color mask  
lower = np.uint8([209, 226, 237])  
upper = np.uint8([255, 255, 255])  
# yellow color mask  
lower = np.uint8([221, 167, 40])  
upper = np.uint8([255, 255, 255])
```



Nó trông khá tốt, ngoại những ảnh ánh sáng yếu bởi cây. Các đường không rõ ràng

YCrCb

Ảnh chuyển từ không gian màu RGB sang YCrCb



In []:

```
# white color mask
lower = np.uint8([197, 0, 118])
upper = np.uint8([255, 255, 255])
# yellow color mask
lower = np.uint8([156, 145, 0])
upper = np.uint8([255, 186, 255])
```



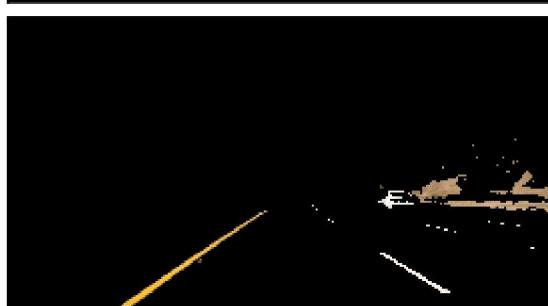
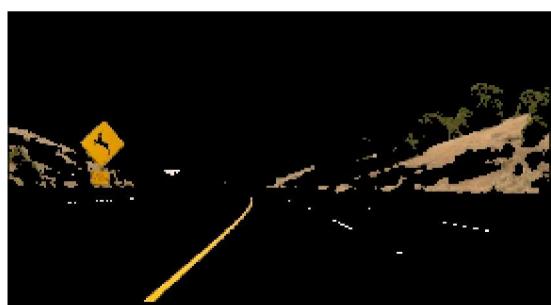
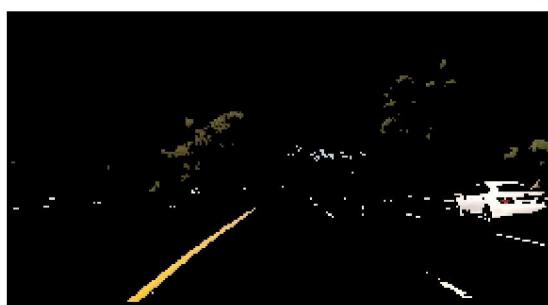
Lab

Ảnh chuyển từ không gian màu RGB sang Lab



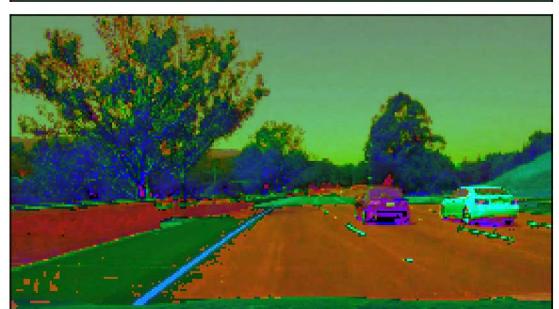
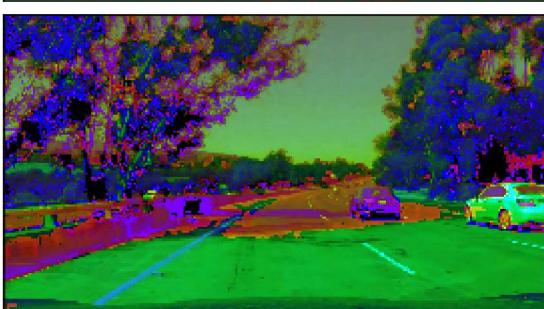
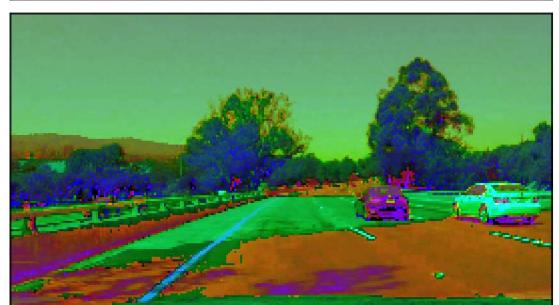
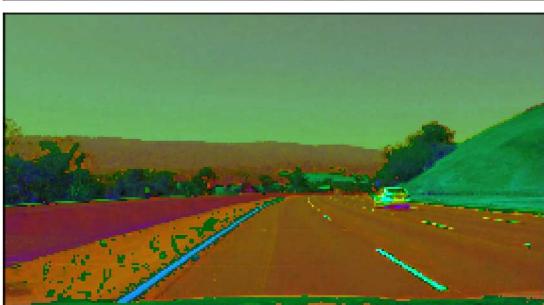
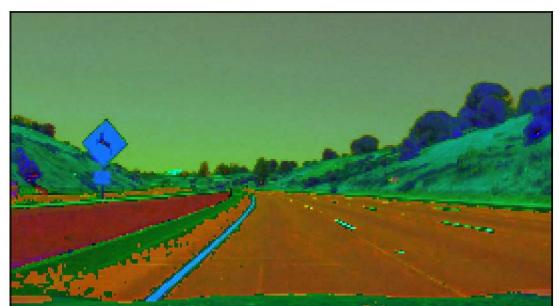
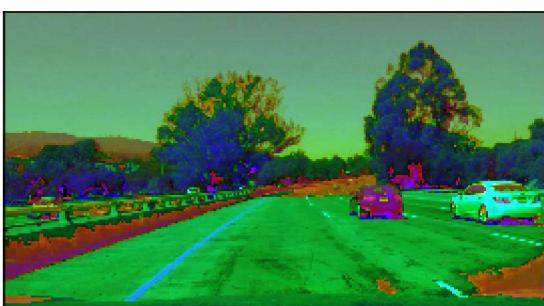
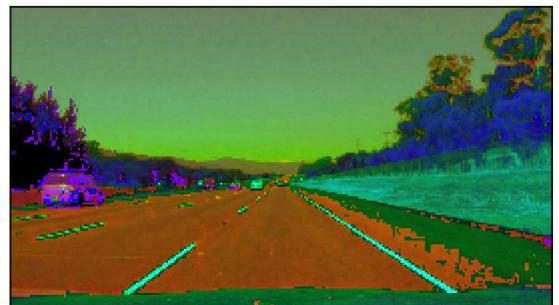
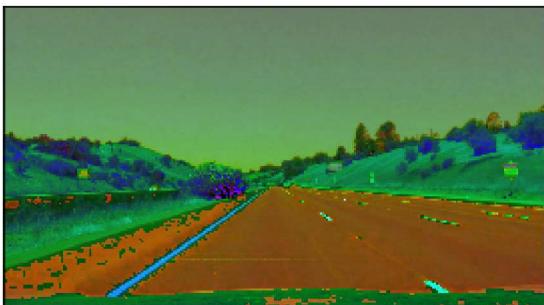
In []:

```
# white color mask
lower = np.uint8([ 214, 0, 104])
upper = np.uint8([255, 255, 255])
white_mask = cv2.inRange(converted, lower, upper)
# yellow color mask
lower = np.uint8([ 71, 55, 151])
upper = np.uint8([ 255, 255, 255])
```



HSL

Ảnh chuyển từ không gian màu RGB sang HSL



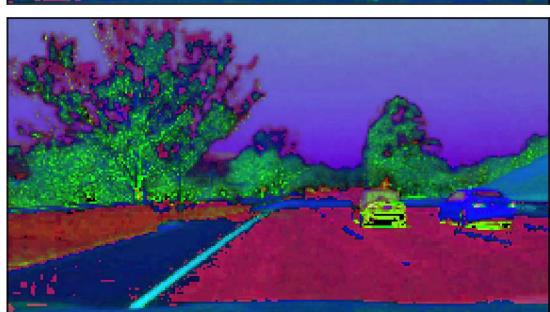
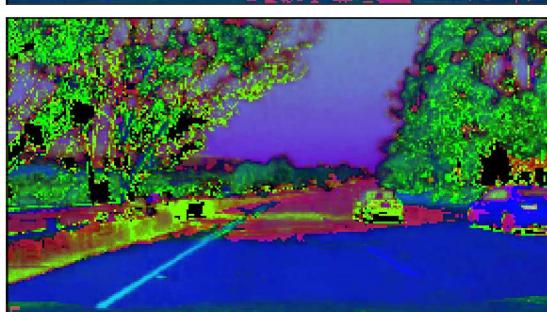
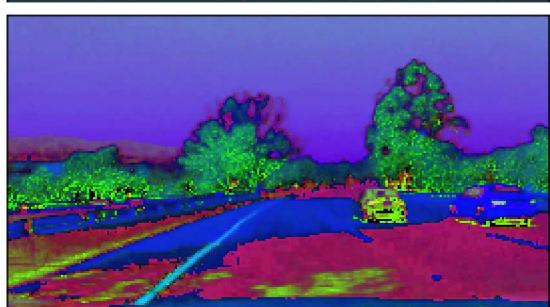
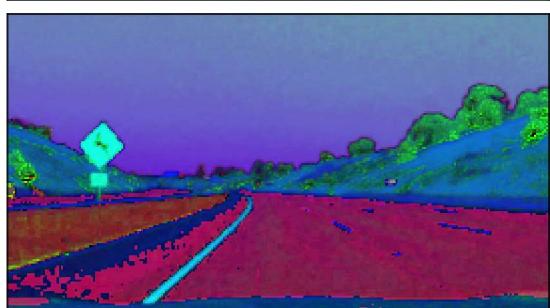
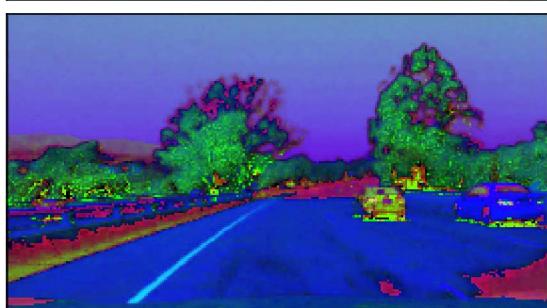
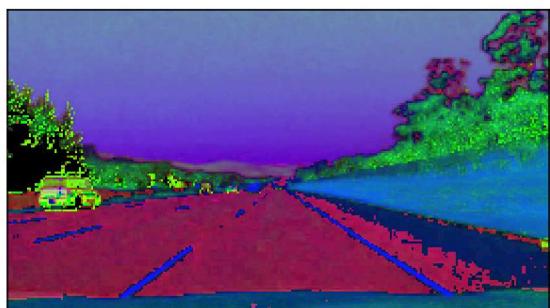
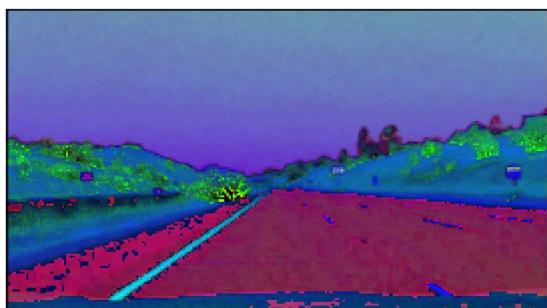
In []:

```
# white color mask
lower = np.uint8([ 0, 200, 0])
upper = np.uint8([255, 255, 255])
# yellow color mask
lower = np.uint8([ 10, 0, 100])
upper = np.uint8([ 40, 255, 255])
```



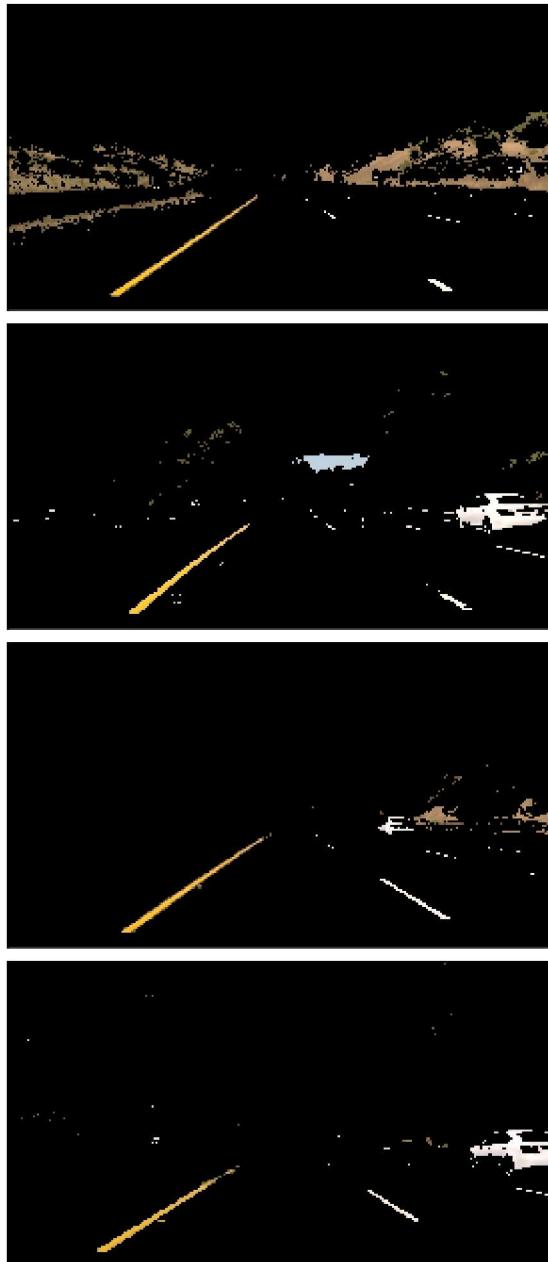
HSV

Ảnh chuyển từ không gian màu RGB sang HSV



In []:

```
# white color mask
lower = np.uint8([0, 0, 216])
upper = np.uint8([255, 39, 255])
# yellow color mask
lower = np.uint8([13, 100, 100])
upper = np.uint8([33, 255, 255])
```



Nhận xét: Đây là data test lý tưởng về ánh sáng lẫn màu sắc xem sự khác biệt rõ trên các khung gian màu
lắng và việc tìm lane cho ra kết quả rất tốt. Nếu ta lấy mẫu data sau:

1



2



3



4



5

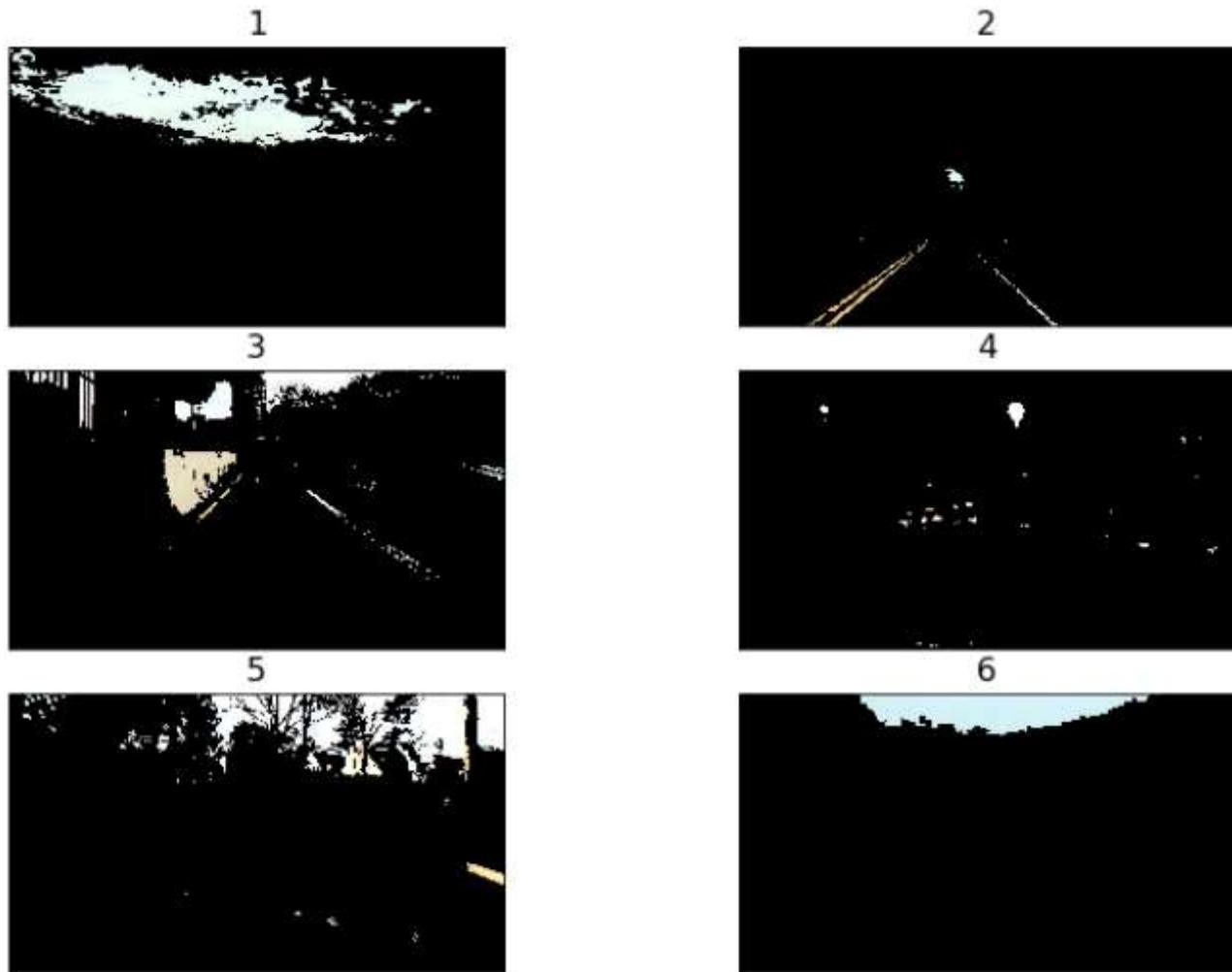


6



sẽ thấy được rằng có rất nhiều vấn đề gặp phải Kết quả lấy ngưỡng với mẫu ảnh này:

RGB



YCrCb

1



2



3



4

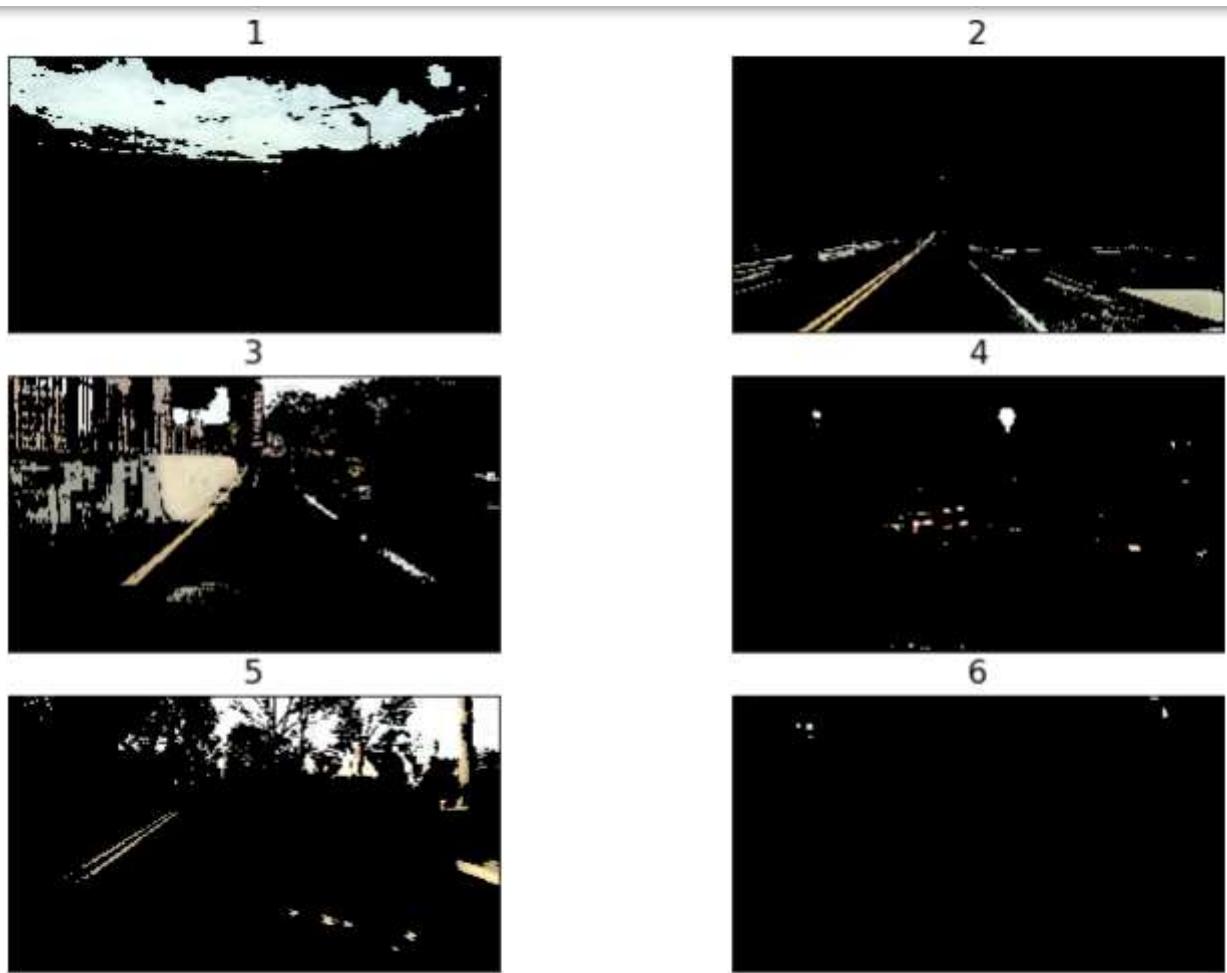


5



6

**Lab**



HSL

1



2



3



4



5



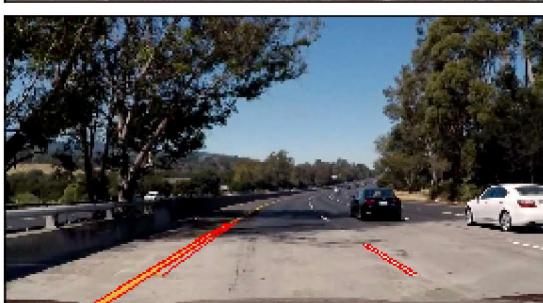
6

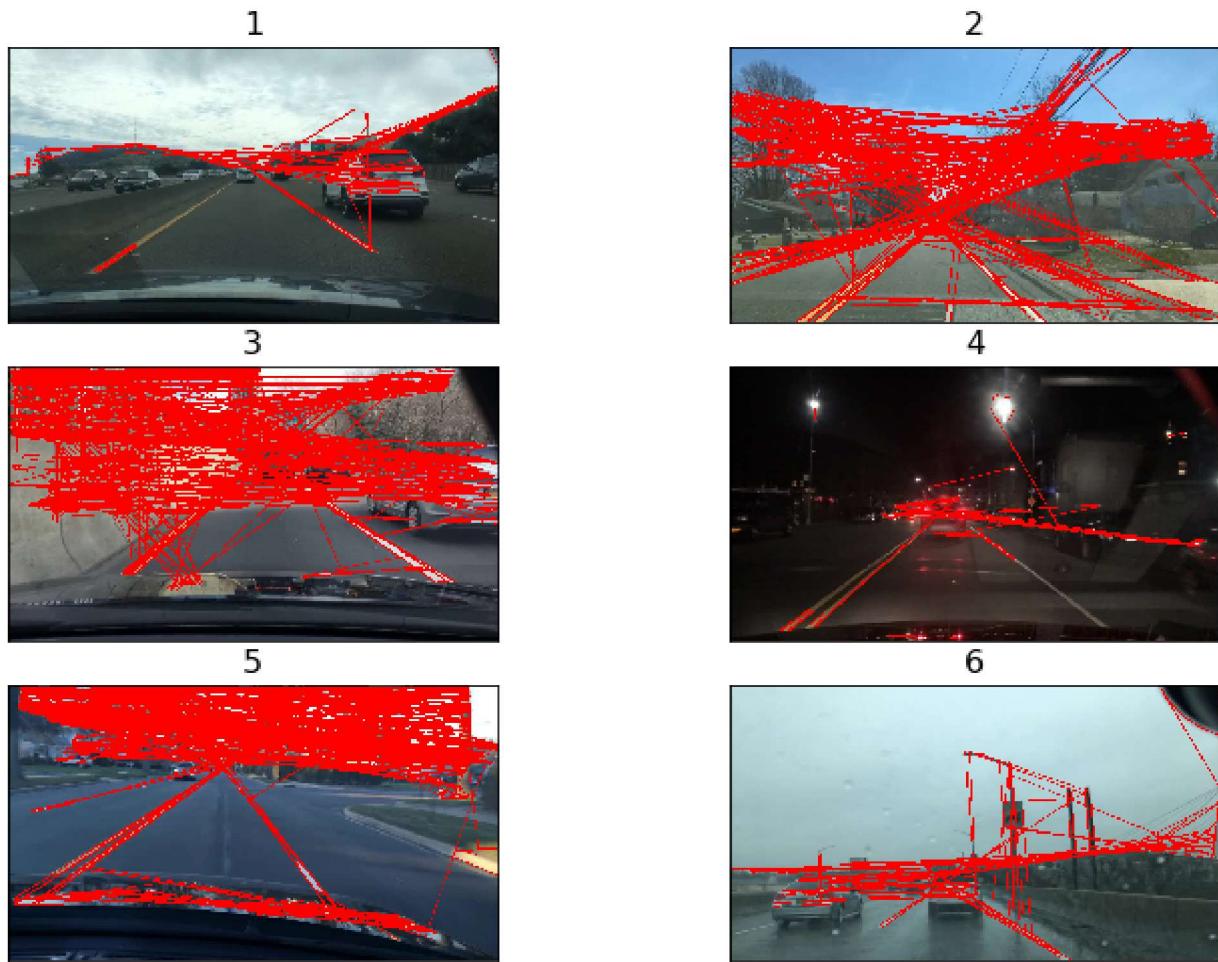


HSV



Để đánh dấu lane có thể dùng kỹ thuật Hough Transform Line ta có kết quả sau (kỹ thuật này đánh dấu các đường thẳng mà ta tìm được):





Kết hợp giữa hai không gian màu

Ta tách từng kênh màu ta có kết quả

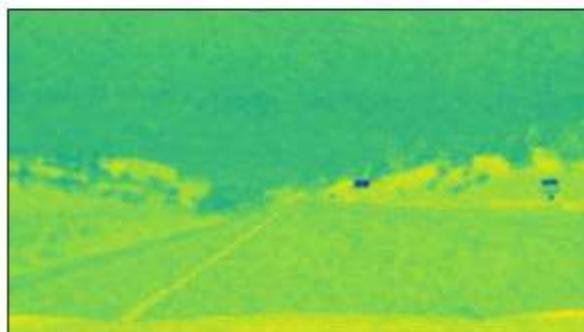
LAB :Nhận thấy kênh B-LAB tốt để tìm lane màu vàng



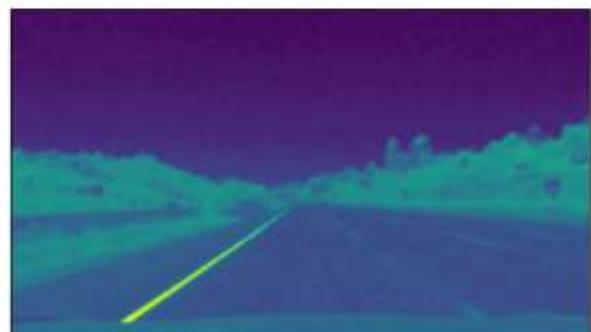
Original LAB image



LAB L-Channel

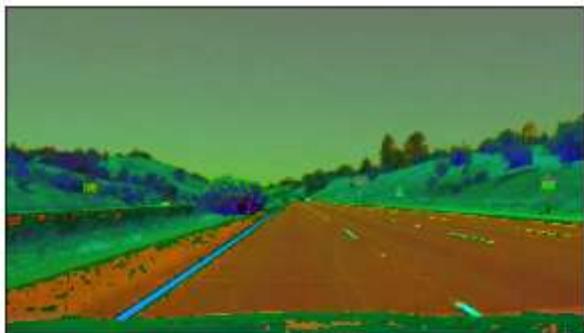


LAB A-Channel

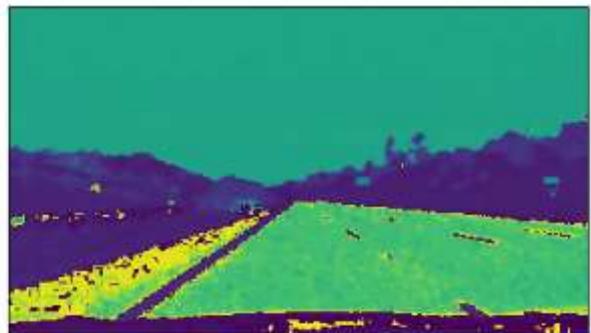


LAB B-Channel

HLS :Nhận thấy kênh L- HLS tốt để tìm lane màu trắng



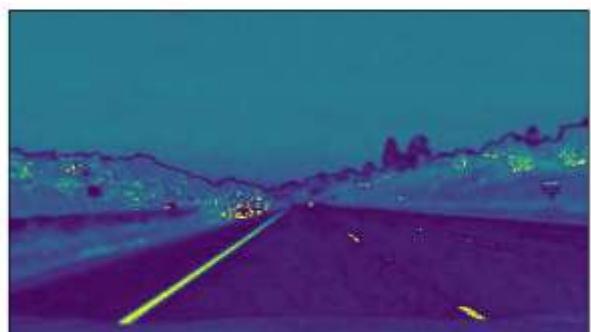
Original HLS image



HLS H-Channel

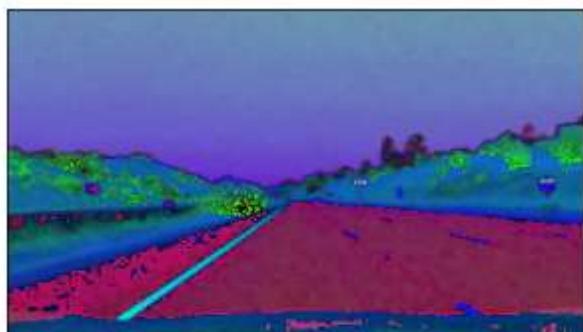


HLS L-Channel



HLS S-Channel

HSV



Original HSV image



HSV H-Channel



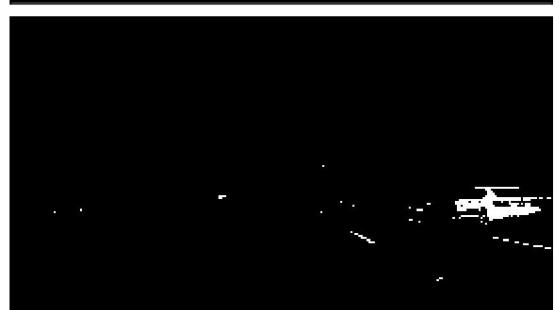
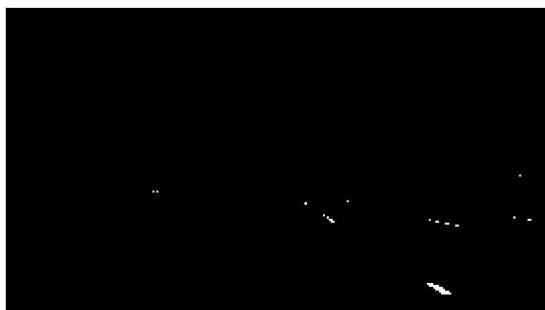
HSV S-Channel



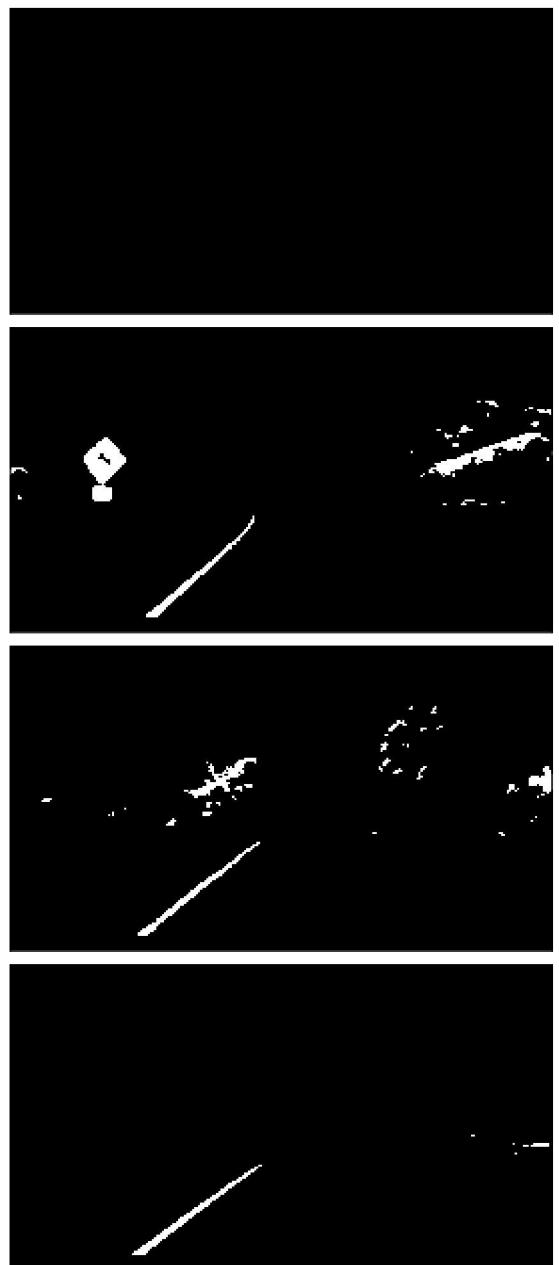
HSV V-Channel

Ở đây dùng kênh L-HLS kết hợp với kênh B-LAB Hoặc cũng có thể kết hợp S-HLS với kênh V-HSV

L-HLS



B-LAB



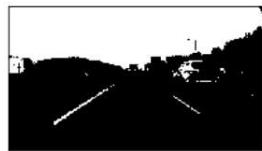
Kết quả kênh L-HLS kết hợp với kênh B-LAB



So sánh kết quả kênh L-HLS kết hợp với kênh B-LAB với data lý tưởng (bên phải) và môi trường (bên trái)



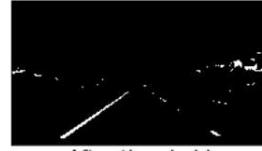
image



After threshold



image



After threshold



image



After threshold



image



After threshold



image



After threshold



image



After threshold



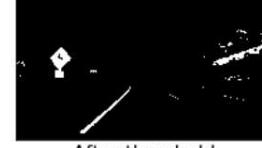
image



After threshold



image



After threshold



image



After threshold



image



After threshold



image



After threshold



image



After threshold

Dùng color transforms và gradients

Ta kết hợp các ưu điểm của không gian màu lại với nhau rồi dùng bộ lọc Sobel để làm tiền xử lý trước khi tìm ra lane.

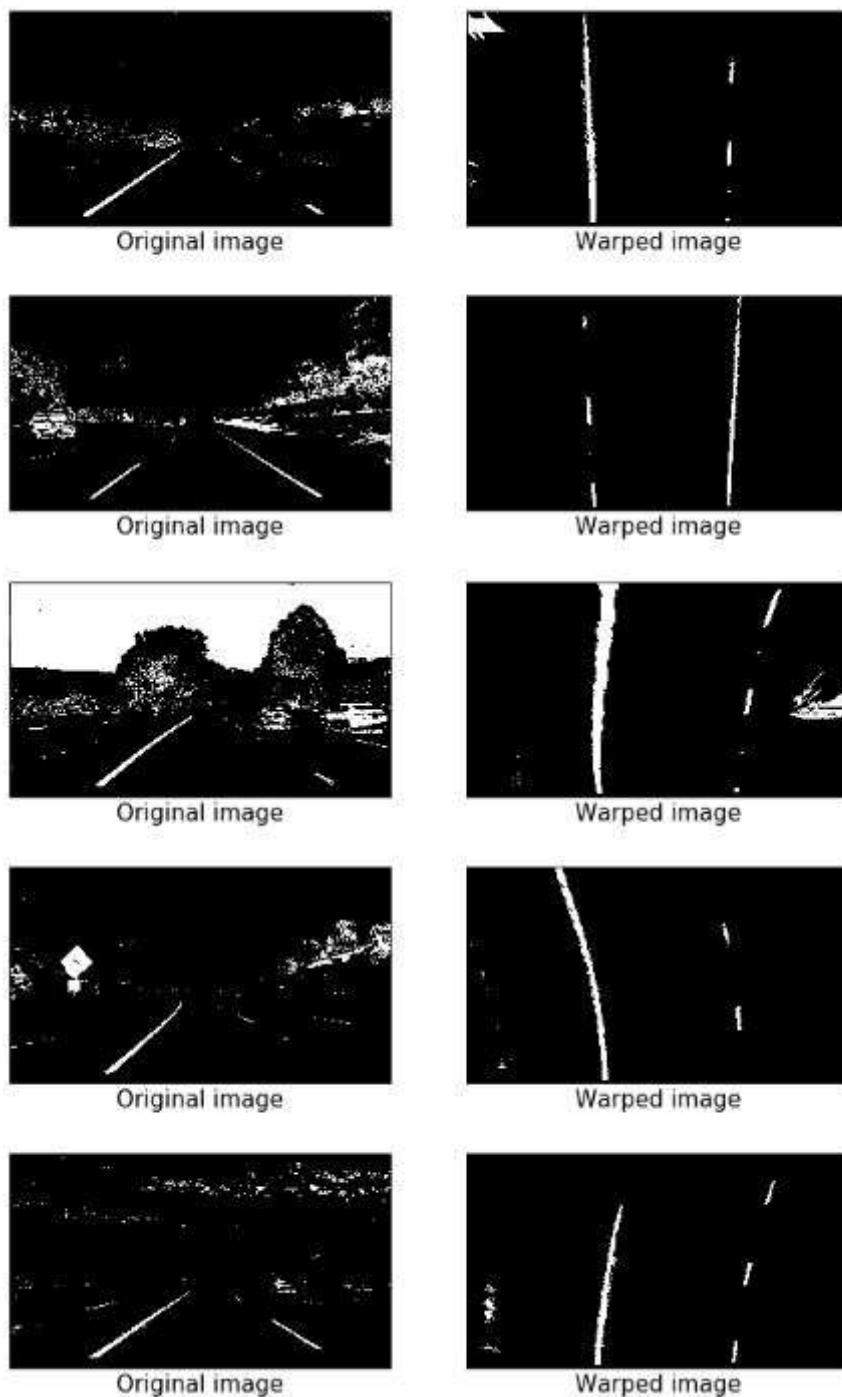


Bước 4: Áp dụng biến đổi phối cảnh để chỉnh sửa hình ảnh nhị phân ("chế độ xem mắt chim").

Thực hiện các biến đổi phối cảnh trước khi lọc bất kì thứ gì, vì biến đổi phối cảnh sẽ loại bỏ phần không mong muốn của hình ảnh và tập trung vào những gì mình muốn lấy. Khi một hình ảnh được chụp bởi một máy ảnh, các vật thể ở xa trông nhỏ hơn trong một hình ảnh. Điều này làm cho các đường song song trong thế giới thực xuất hiện như thể chúng giao nhau trong hình ảnh. Để hoàn tác một hiệu ứng như vậy, chúng ta cần chuyển đổi hình ảnh thành một hình ảnh khác. Chuyển đổi phối cảnh giúp xem cảnh từ điểm quan sát khác (như máy ảnh được di chuyển một số vị trí khác và ảnh được chụp). Vì vậy, chúng tôi quyết định xem con đường từ một điểm ngắm chim. Để thực hiện chuyển đổi phối cảnh, đã chọn bốn điểm trong một hình ảnh chưa được phân loại ban đầu và sau đó ánh xạ nó thành một tập hợp gồm bốn điểm tạo thành một hình chữ nhật.

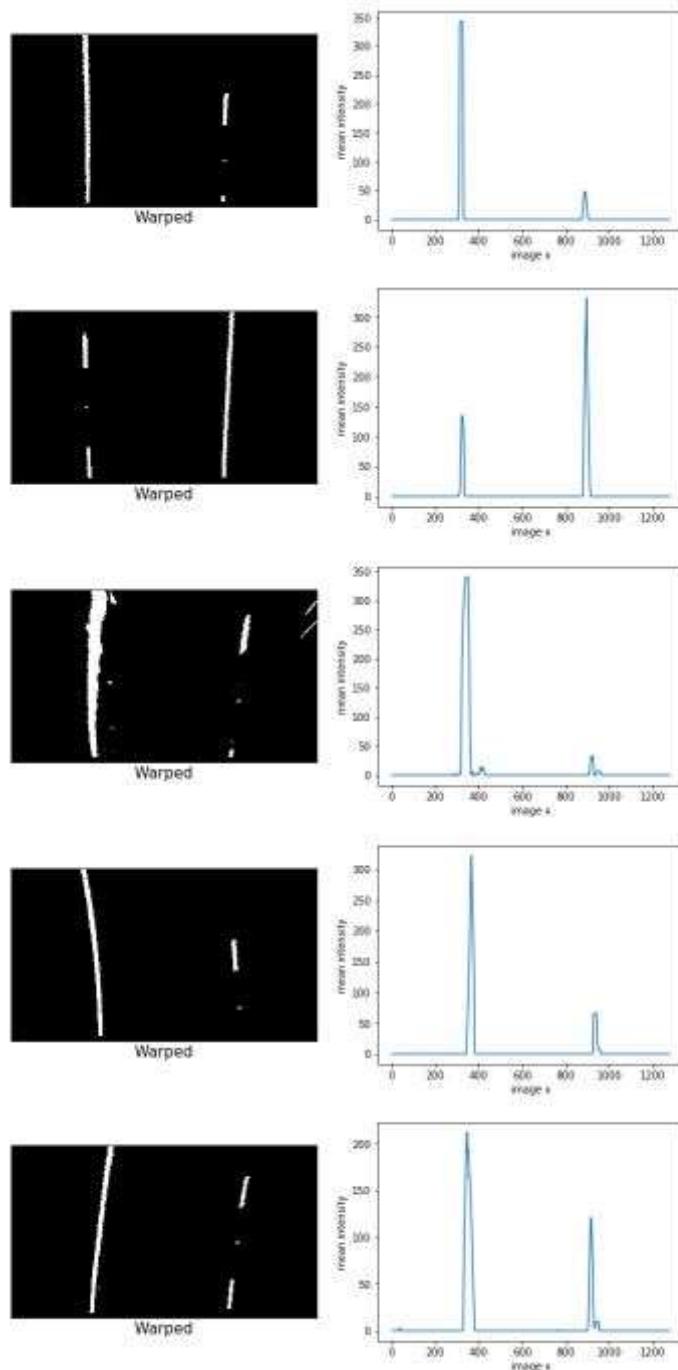


Sử dụng `getPerspectiveTransform (src, dst)` để lấy ma trận phối cảnh (M). Sau đó, chúng tôi sử dụng ma trận M và `warpPerspective (undist, M, img_size)` để tính toán hình ảnh được chuyển đổi phối cảnh. `Img_size` phải ở dạng cột, hàng

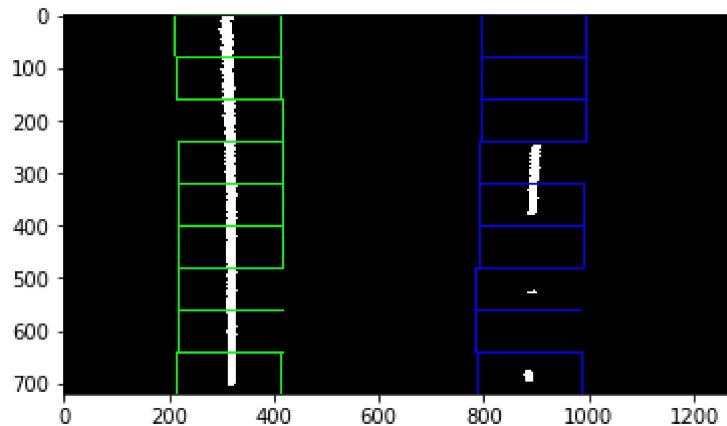


Bước 5: Phát hiện pixel làn đường và phù hợp để tìm ranh giới làn đường..

Để xác định lane trái lane phải ta thực hiện như sau: Bước đầu tiên là tạo một biểu đồ Histogram hình ảnh. Bằng cách này, chúng tôi có thể tìm ra sự khác biệt giữa pixel làn đường bên trái và pixel làn đường bên phải.



Từ đây ta có thể định nghĩa được lane trái và phải thông qua biểu đồ mật độ điểm ảnh. Xong để đánh dấu cũng như để lane được tìm thấy liên tục không bị đứt quãng ta dùng kỹ thuật SlidingWindow. Dưới đây là kết quả khi dùng slide window



The next step is to initiate a Sliding Window Search in the left and right parts which we got from the histogram.

The sliding window is applied in following steps:

1. The left and right base points are calculated from the histogram
2. We then calculate the position of all non zero x and non zero y pixels.
3. We then Start iterating over the windows where we start from points calculate in point 1.
4. We then identify the non zero pixels in the window we just defined
5. We then collect all the indices in the list and decide the center of next window using these points
6. Once we are done, we separate the points to left and right positions
7. We then fit a second degree polynomial using np.polyfit and point calculate.

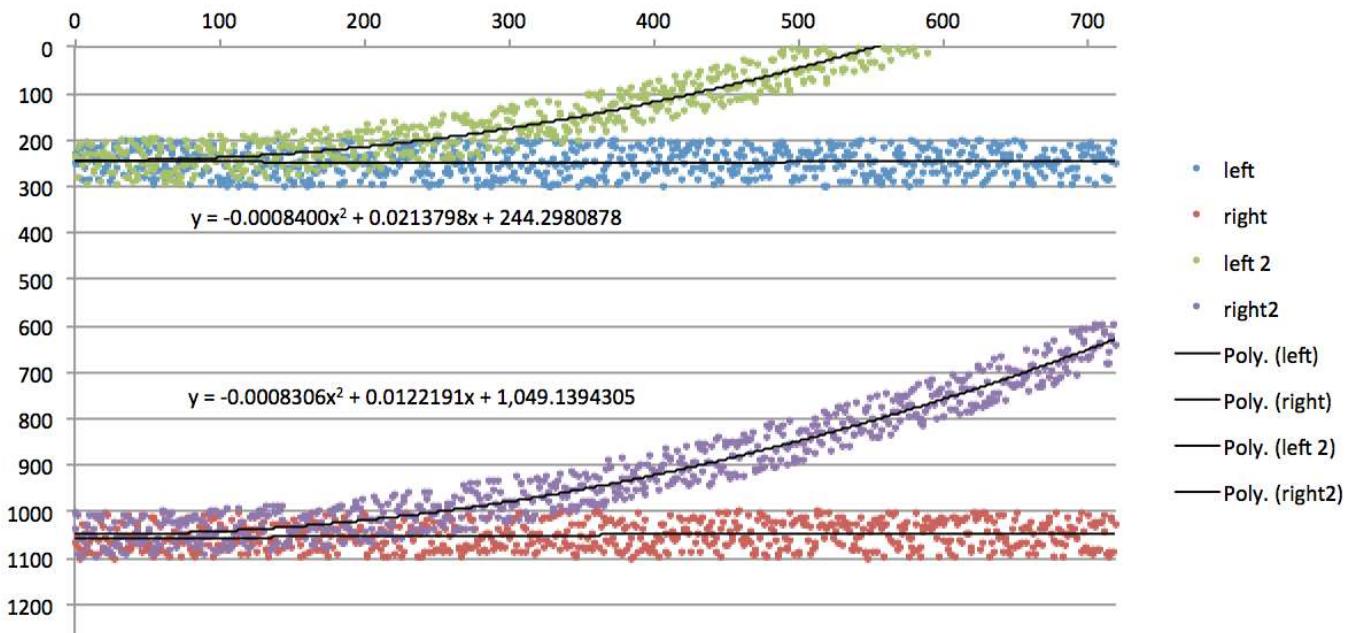
Bước 6: Xác định độ cong của làn đường và vị trí của xe đối với trung tâm.

Để tính góc Radius:

- Đầu tiên chúng ta định nghĩa các giá trị để chuyển đổi pixel thành mét
- Vẽ đường lane trái và phải
- Tính toán độ cong từ làn đường trái và phải một cách riêng biệt
- Trả về giá trị trung bình của các giá trị được tính trong bước 3.

Đối với khoảng cách Chúng ta biết rằng trung tâm của hình ảnh là trung tâm của chiếc xe. Để tính toán độ lệch từ trung tâm, chúng ta có thể quan sát vị trí pixel ở làn bên trái và làn đường bên phải. Lấy giá trị trung bình của điểm bên trái dưới cùng bên trái của làn bên trái và dưới cùng bên phải điểm cuối cùng của làn đường bên phải và sau đó trừ nó khỏi trung tâm của chiếc xe để có được độ lệch từ trung tâm.

Vẽ các đường làn trên như hình ảnh, xác định bán kính cong của các đường và xác định vị trí của ô tô so với tâm của làn đường. Các đa thức phù hợp có dạng:



Cách tính góc

$$f(y) = Ay^2 + By + C$$

Từ phương trình này, bán kính cong có thể được tính tại bất kỳ điểm nào x của hàm $x = f(y)$ được đưa ra như sau:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{\left| \frac{d^2x}{dy^2} \right|}$$

$$f'(y) = \frac{dx}{dy} = 2Ay + B$$

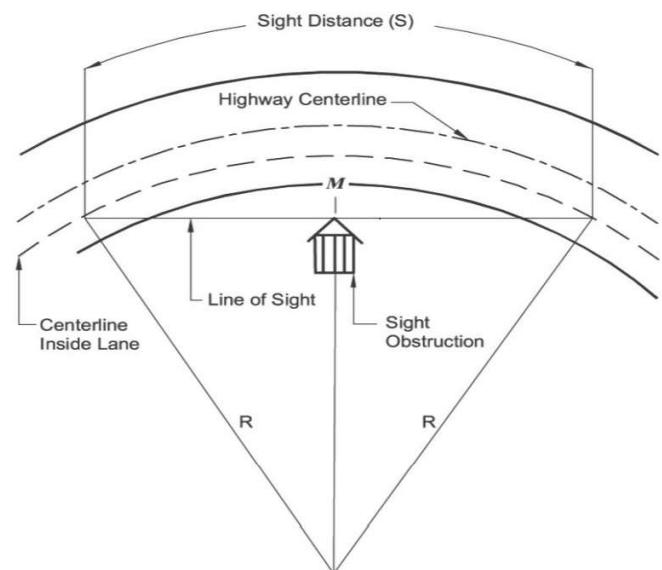
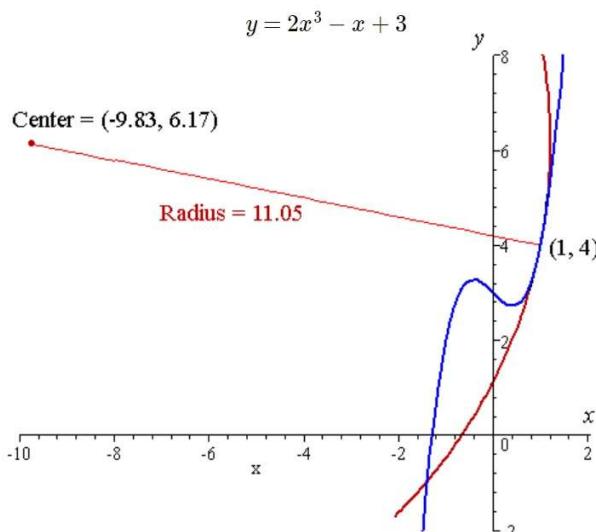
$$f''(y) = \frac{d^2x}{dy^2} = 2A$$

Vì vậy, phương trình bán kính cong sẽ trở thành:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

Bán kính cong được tính toán lần đầu tiên trong không gian pixel , nhưng chúng ta thực sự cần phải lặp lại phép tính này sau khi chuyển đổi các giá trị x và y của chúng tới không gian thực . Điều này được thực hiện bằng cách nhân x và y theo khoảng cách từ thực . 'Làn đường được giả định dài khoảng 30 mét và rộng 3,7 mét.'

Vị trí của tương đối của xe so với trung tâm có thể được tìm thấy bằng cách xem xét sự khác biệt của trung tâm của hình ảnh the center of the lane (right lane + left lane / 2) . Điều này cũng được chia tỷ lệ trong không gian thế giới thực.



Tham khảo cách chứng minh ở đây:

- [\(https://www.intmath.com/applications-differentiation/8-radius-curvature.php\)](https://www.intmath.com/applications-differentiation/8-radius-curvature.php)