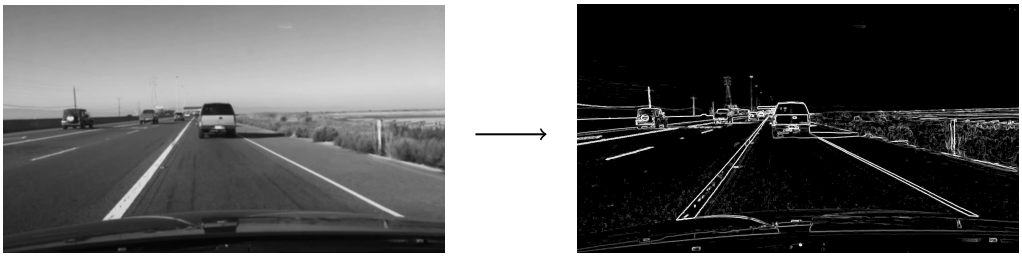
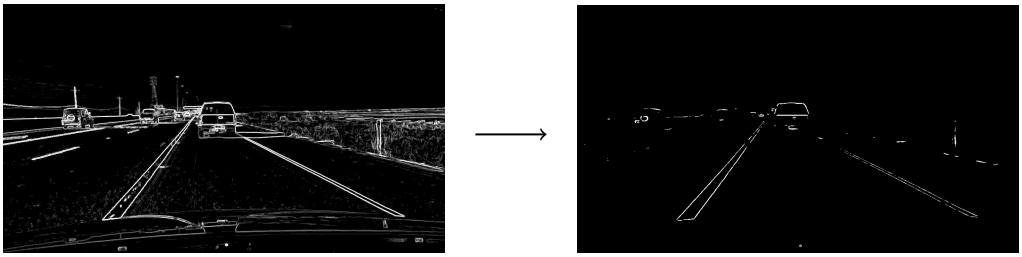
example of this, using a Gaussian blur with $K_x = K_y = 5$ and $\sigma = 2.1$ beforehand, can be seen in *Figure 25* . The difference is noticeable and because of this, although the filter is computationally expensive, I have chosen to implement the blur.



$K_x = K_y = 5$
$\sigma = 2.1$

Figure 25: A fully preprocessed test image subjected to the Sobel-Feldman operator

If we apply a threshold with an intensity of $T_G = 80\%$ to the output of the Sobel operator seen in *Figure 25* we can discard a lot of artifacts from the image. The result of this operation can be seen in *Figure 26* . The lane markings persist, but many of the details in the roadside get discarded.



$T_G = 80\%$

Figure 26: Example of thresholding applied to the output of the Sobel-Feldman operator

4.2.3 k-means Image Segmentation Implementation

When applying the Gaussian and Sobel steps as noted in *4.2.2 Filter Development* , I still noticed a lot of noise in the Sobel output. As recommended by my mentor, I took a step back and implemented a filter that could separate the pixels in a grayscale image using k-means. The intention of this filter is to separate the lane from the non-lane pixels.

I implemented the filter using Lloyd's algorithm. For each pixel, the program keeps the luminance value and the distance to the closest centroid. Each centroid has a mean luminance value that gets refined every iteration so that it nears the real mean value. After S_i iterations of the algorithm, the approximate means for all clusters are calculated. Then, each pixel's luminance value gets updated to the nearest cluster's luminance value. The memory required for this approach is $4 \cdot I_w \cdot I_h$ bytes because every pixel takes 4 bytes: two for storing the nearest luminance value, one for storing its own luminance value and one for storing the cluster ID. Therefore, we could easily store it in FPGA registers.