I haven’t gotten very far in the project as I just started this weekend, so far my personal investigation has only been implementing the image processing algorithms presented in class. I decided to implement my program in Java for the built in graphics libraries so that I can easily manipulate the images. The following descriptions and images are demonstrations of the program in its current state.

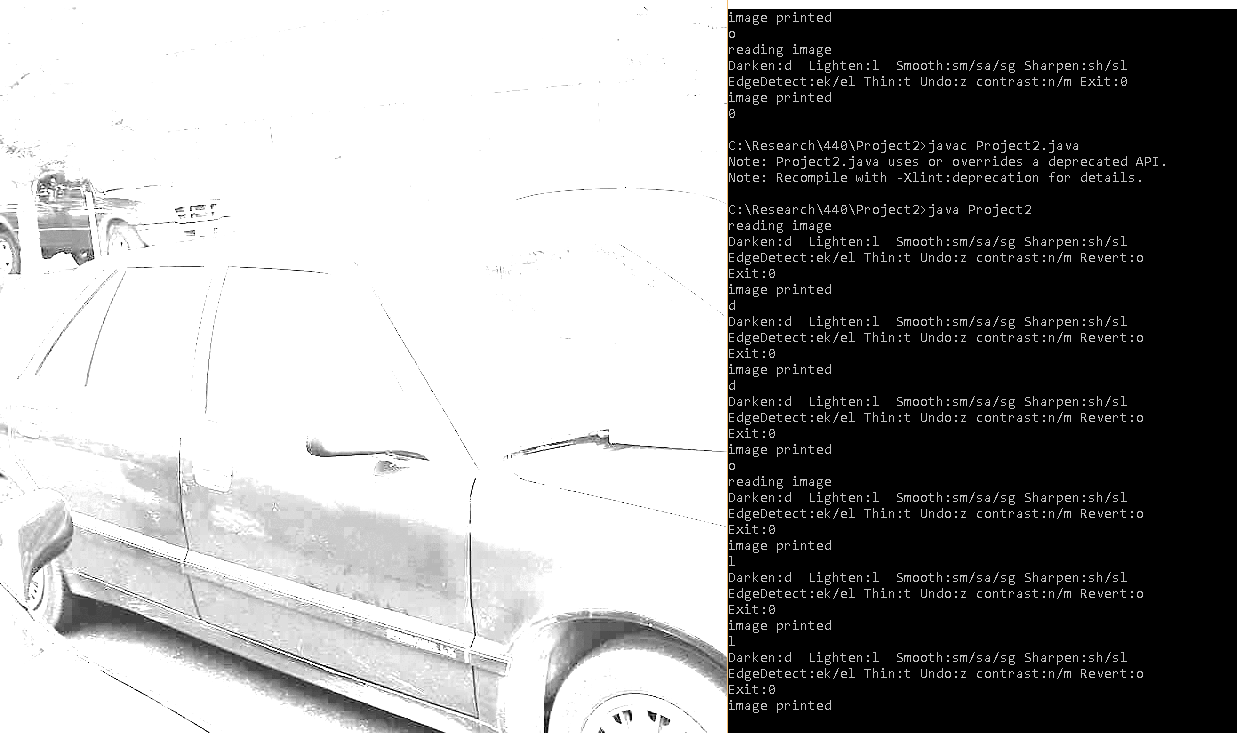
Original image (image 1 to detect box in)



* **Brightness**

I uniformly increase or decrease pixel values by a specific amount (add/subtract), if it goes below 0 then it is 0, and if it goes above 255 then it is 255.

Increase Brightness



Decrease brightness



* **Contrast**

To adjust contrast I pick a middle value and a degree in which to shift away from the middle value and move each pixel away from that value by adding or subtracting the chosen value. Pixels that are close to the middle value are unaltered (within 5 light values). In the picture below the middle value was chosen to be 100.

Increase Contrast



Decrease Contrast



* **Smoothing** 
  + Averaging method

I simply make the middle pixel of a 3x3 pixel square the average of all of the values in the square (if at the image boundaries then only some of the pixels contribute to the average)



* + Approximate Gaussian smoothing is also implemented using the same method with the following weights: and dividing it by 16



* + Median method

I implemented the median method using a quick median approximation method described in the paper :

An Efficient Algorithm for the Approximate Median Selection Problem

by S. Battiato, D. Cantone, D. Catalano and G. Cincotti from University of Catania, Italy and M. Hofri from Worcester MA.



* **Sharpening**
  + Histogram Equalization method

Using the algorithm presented in class:



* + Laplacian sharpening

I used the convolution method on the following two kernels.

* + - Kernel 1: the second kernel presented in class:

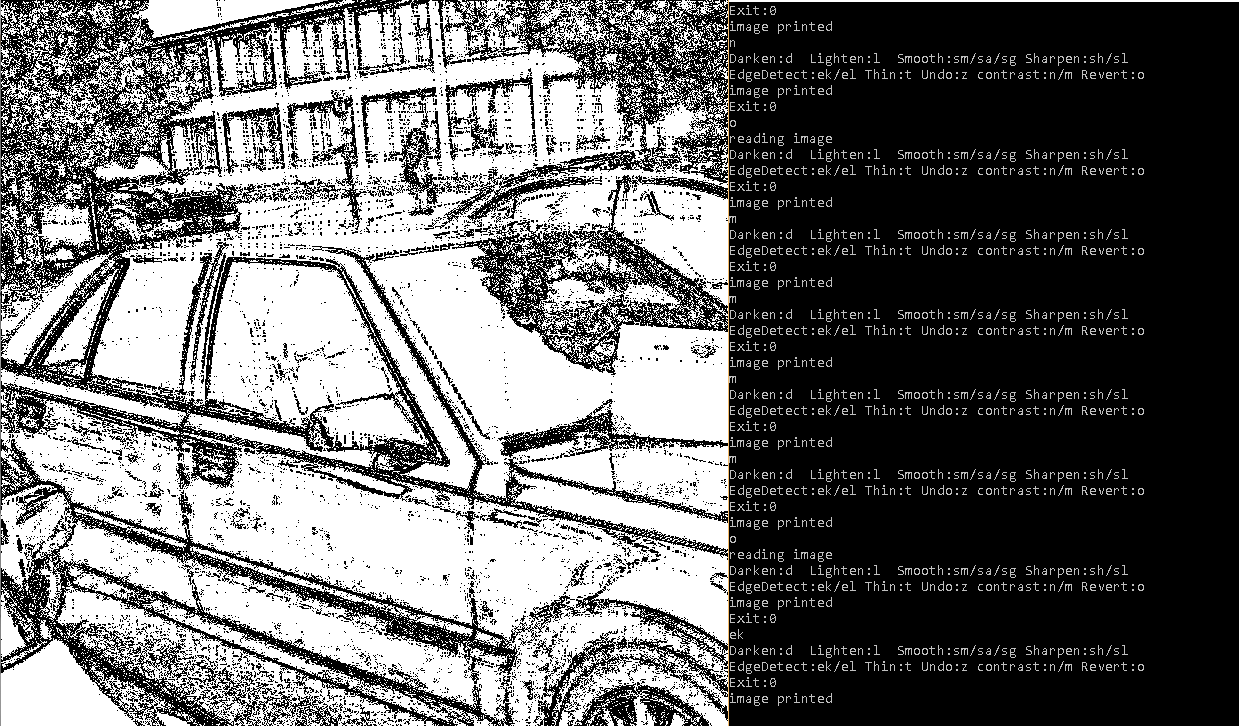


* + - Kernel 2: found online



* **Edge Detection**
  + Edge detection using Kirsch Operators

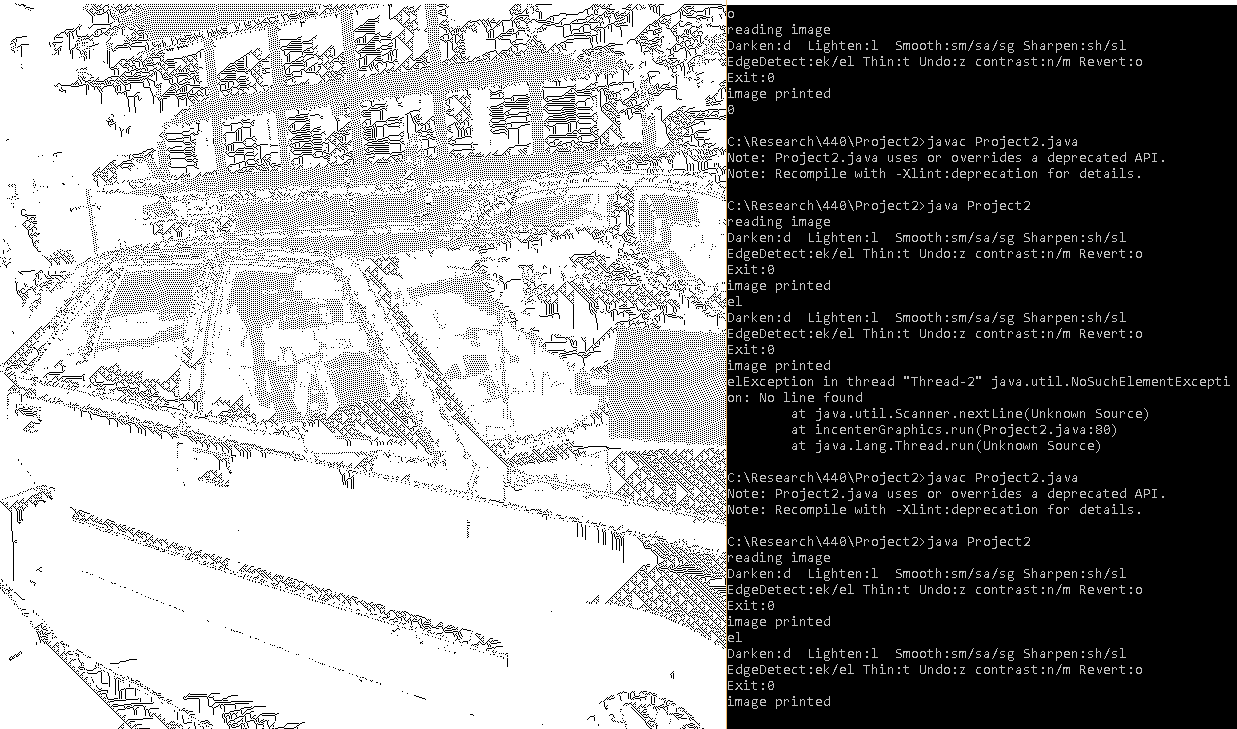
I specify how close to 0 the differences have to be for a pixel to be marked as an edge element. If a pixel is determined to be an edge element by any of the kirsch operators then it is an edge element. The resulting image is an image of only edge elements.



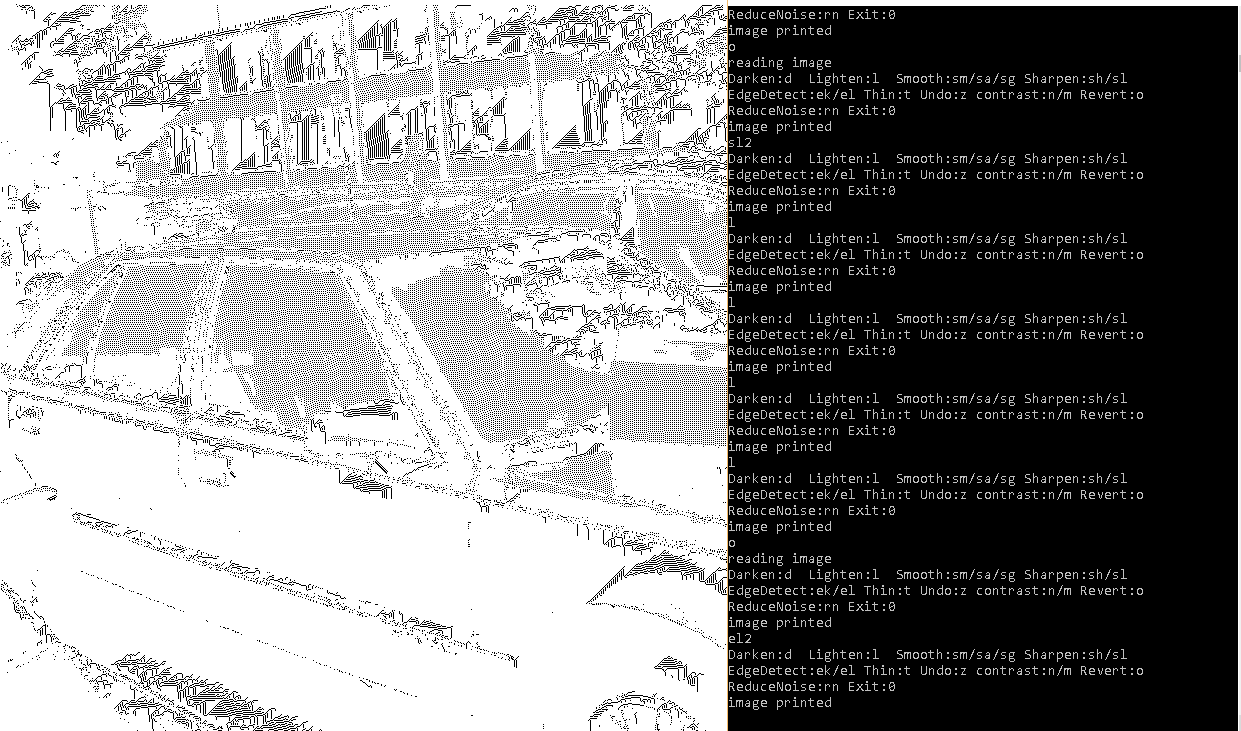
* + Edge detection using Laplacian

Very similar to the sharpening Laplacian, if the convolution results in a value that is close enough to zero then the pixel is marked as an edge element. This is done with the same two kernels as above.

* + - Kernel 1: the second kernel presented in class:



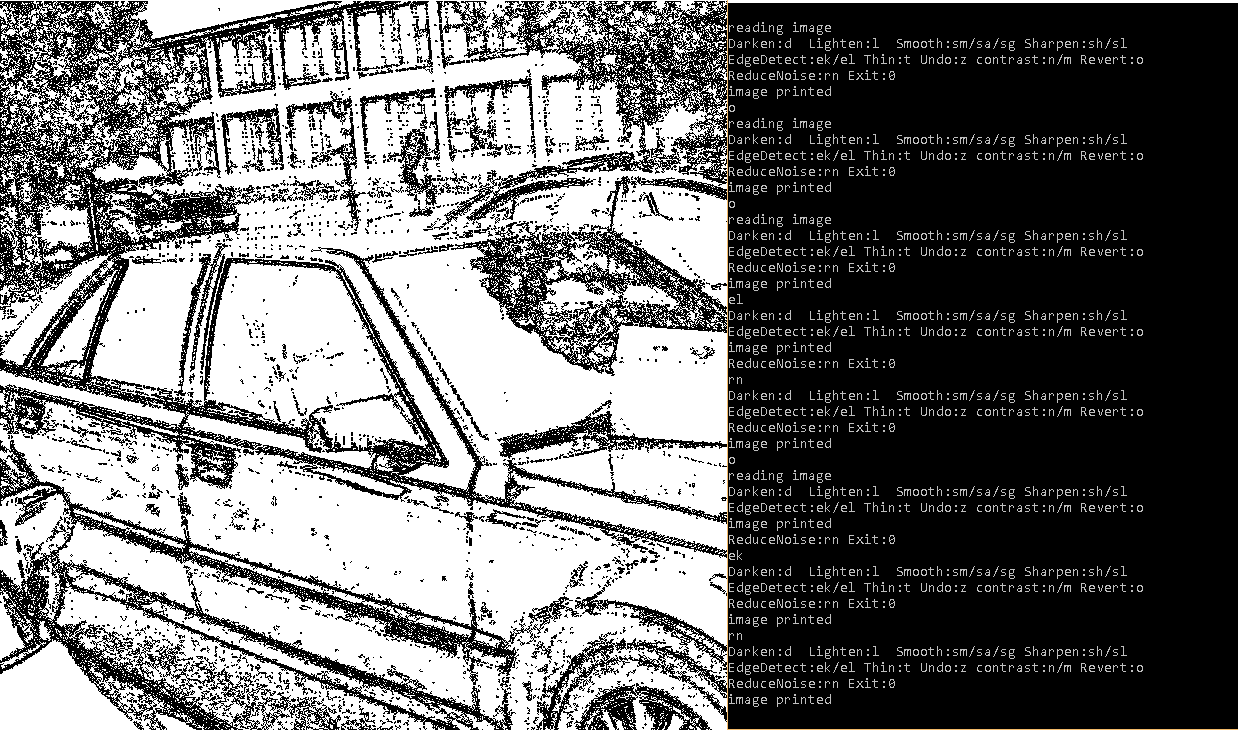
* + - Kernel 2: found online



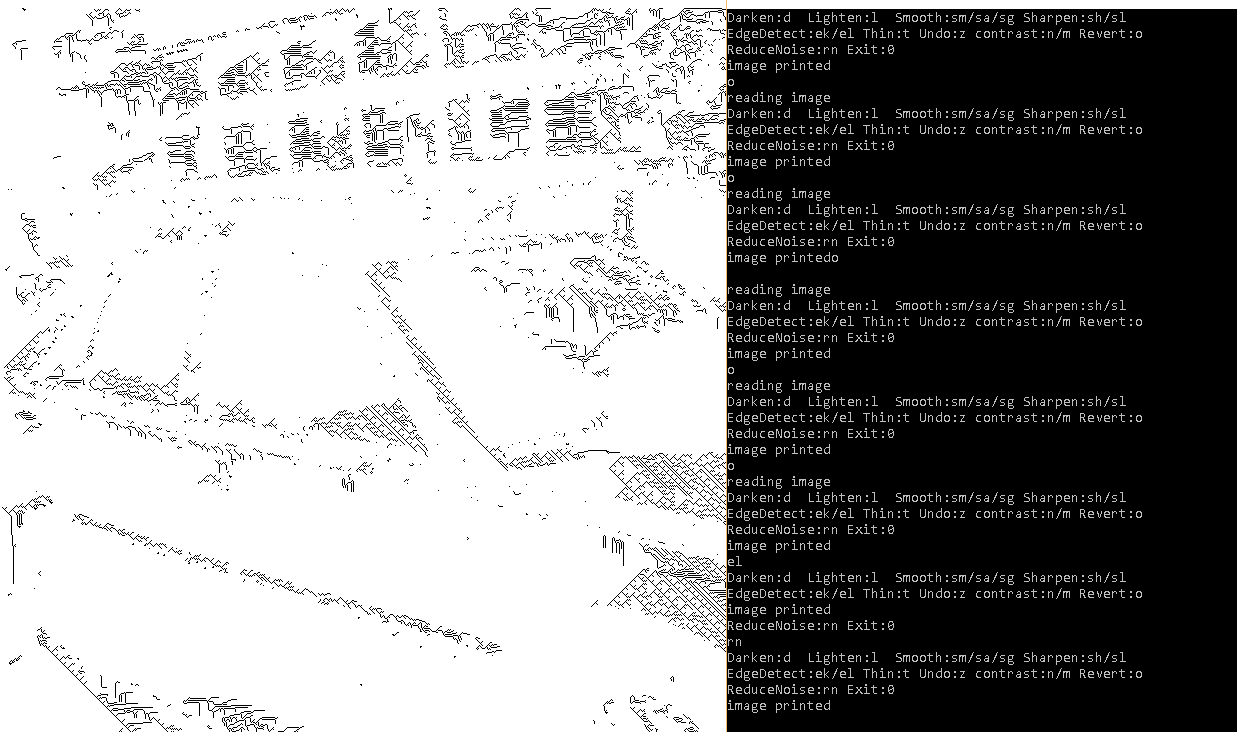
* **Noise Reduction**

To reduce noise, I count the amount of neighboring pixels in the 3x3 window about a single pixel. If that pixel has less than a specified amount of neighboring black pixels then that pixel is changed to white.

Noise reduction on edge dection (kirsch operators)

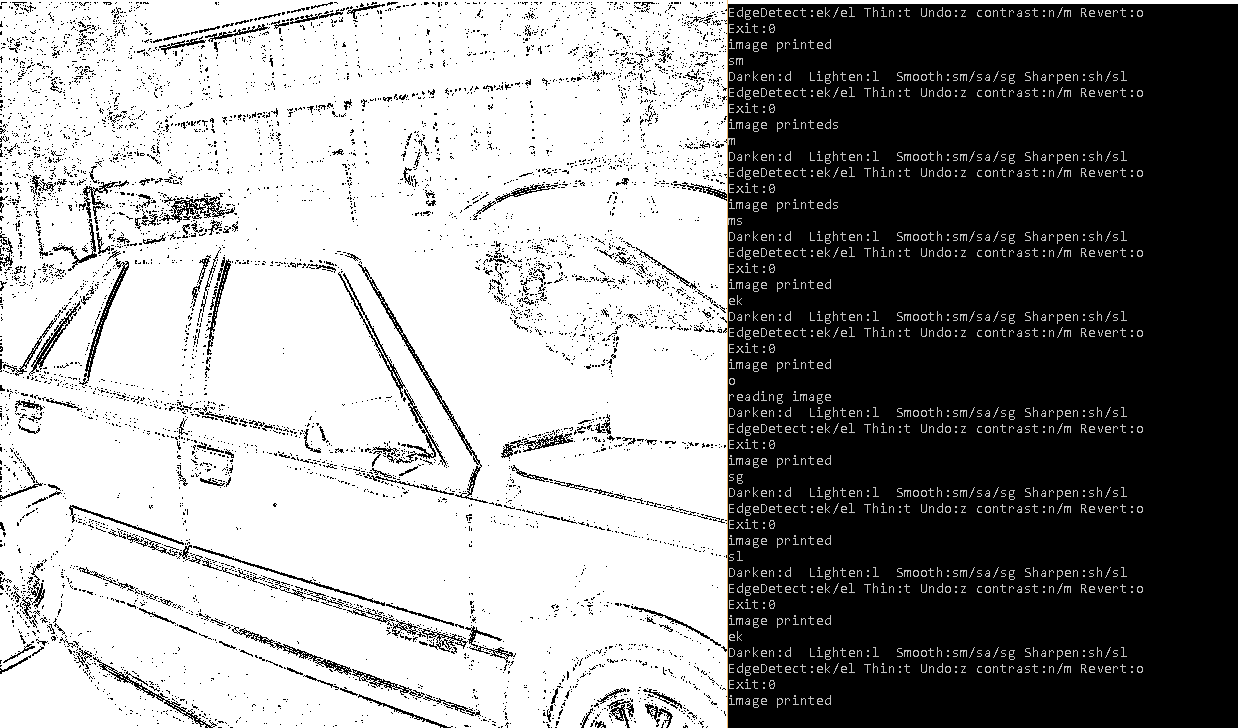


Noise reduction on laplacian edge detection on kernel 1



* **Thinning**

I implemented the Stefanelli and Rosenfeld algorithm described in class. It seems to work well. I am concerned about my edges having white specs in them which makes the thinning algorithm not outline shapes completely. This can probably be addressed by altering parameters and doing combinations of above image processing operations.



* **Other: Saving/undo/revert**

I also implemented ways to save the current image, undo changes to the image and revert to the original.

(left) Smoothed gaussian, sharpened and kirch edge detect and thinned

(right) an interesting combination

