Fundamentals of Computer Vision

Project Assignment 1
Image Filtering and Hough Transform

In this assignment you will be implementing some basic image processing algorithms and putting them together to build a Hough Transform based line detector. Your code will be able to find the start and end points of straight line segments in images. We have included a number of images for you to test your line detector code on. Like most vision algorithms, the Hough Transform uses a number of parameters whose optimal values are (unfortunately) data dependent, that is, a set of parameter values that works really well on one image might not be best for another image. By running your code on the test images you will learn about what these parameters do and how changing their values affects performance.

Many of the algorithms you will be implementing as part of this assignment are functions in the Matlab image processing toolbox. You are not allowed to use calls to functions in this assignment. You may however compare your output to the output generated by the image processing toolboxes to make sure you are on the right track.

Instructions

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. **Start early!** Especially those not familiar with Matlab.

3. **Questions:** If you have any questions, please look at canvas discussion first. Other students may have encountered the same problem, and it may be solved already. If not, post your question on the discussion board. Teaching staff will respond as soon as possible.

4. **Write-up:** Your write-up should mainly consist of three parts, your answers to theory questions, the resulting images of each step, that is, the output of `houghScript.m`, and the discussions for experiments. Please note that we **DO NOT** accept handwritten scans for your write-up in this project. Please type your answers to theory questions and discussions for experiments electronically.

5. **Submission:** Your submission for this assignment should be a zip file, <First_FmilyName.zip>, composed of your write-up, your Matlab implementations (including any helper func-tions), and your implementations, results for extra credit (optional). Please make sure

to remove the `data/` and `result/` folders, the `houghScript.m` and `drawLine.m` scripts, and any other temporary files you generated.

Your final upload should have the files arranged in this layout:

<First_FmilyName>.zip

- <First_FmilyName>
  - <First_FmilyName>.pdf
  - matlab
    - myImageFilter.m
    - myEdgeFilter.m
    - myHoughTransform.m
    - myHoughLines.m
    - any helper functions you need
  - ec
    - myHoughLineSegments.m
    - ec.m
    - your own images
    - your own results

6. **File paths:** Please make sure that any file paths that you use are relative and not absolute. Not `imread('/name/Documents/subdirectory/hw1/data/xyz.jpg')` but `imread('../data/xyz.jpg')`.

1 Theory questions

Type down your answers for the following questions in your write-up. Each question should only take a couple of lines. In particular, the proofs" do not require any lengthy calculations. If you are lost in many lines of complicated algebra you are doing something much too complicated (or wrong).

Q1.1 Hough Transform Line Parametrization (20 points)

1. Show that if you use the line equation $\rho = x\cos\theta + y\sin\theta$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y).
2. Why do we parametrize the line in terms (ρ, θ) instead of the slope and intercept (m, c)? Express the slope and intercept in terms of (ρ, θ).
3. Assuming that the image points (x, y) are in an image of width W and height H, that is, $x \in [1, W]$, $y \in [1, H]$, what is the maximum absolute value of ρ, and what is the range for θ?

4. For point $(10, 10)$ and points $(20, 20)$ and $(30, 30)$ in the image, plot the corresponding sinusoid waves in Hough space, and visualize how their intersection point defines the line. What is (m, c) for this line?. Please use Matlab to plot the curves and report the result in your write-up.

2 Implementation

We have included a main script named houghScript.m that takes care of reading in images from a directory, making function calls to the various steps of the Hough transform (the functions that you will be implementing) and generates images showing the output and some of the intermediate steps. You are free to modify the script as you want, but note that TAs will run the original houghScript.m while grading. Please make sure your code runs correctly with the original script and generates the required output images.

Every script and function you write in this section should be included in the `matlab/` directory. Please include resulting images in your write-up.

Q2.1 Convolution (20 points)

Write a function that convolves an image with a given convolution filter

```
function [img1] = myImageFilter(img0, h)
```

As input, the function takes a greyscale image (`img0`) and a convolution filter stored in matrix `h`. The output of the function should be an image `img1` of the same size as `img0` which results from convolving `img0` with `h`. You can assume that the filter `h` is odd sized along both dimensions. You will need to handle boundary cases on the edges of the image. For example, when you place a convolution mask on the top left corner of the image, most of the filter mask will lie outside the image. One solution is to output a zero value at all these locations, the better thing to do is to pad the image such that pixels lying outside the image boundary have the same intensity value as the nearest pixel that lies inside the image.

You can call Matlab's function to pad array. However, your code can not call on Matlab's `imfilter`, `conv2`, `convn`, `filter2` functions, or any other similar functions. You may compare your output to these functions for comparison and debugging. It is better to vectorize this function. Examples and meaning of vectorization can be found here. Specifically, try to reduce the number of `for` loops that you use in the function as much as possible.

Q2.2 Edge detection (20 points)

Write a function that finds edge intensity and orientation in an image. Display the output of your function for one of the given images in the handout.

```
function [img1] = myEdgeFilter(img0, sigma)
```

The function will input a greyscale image (`img0`) and scalar (`sigma`). `sigma` is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. The function will output `img1`, the edge *magnitude* image.

First, use your convolution function to smooth out the image with the specified Gaussian kernel. This helps reduce noise and spurious fine edges in the image. Use `fspecial` to get the kernel for the Gaussian filter. The size of the Gaussian filter should depend on `sigma` (e.g., `hsize = 2 * ceil(3 * sigma) + 1`).

The edge magnitude image `img1` can be calculated from image gradients in the x direction and y direction. To find the image gradient `imgx` in the x direction, convolve the smoothed image with the x-oriented Sobel filter. Similarly, find image gradient `imgy` in the y direction by convolving the smoothed image with the y-oriented Sobel filter. You can also output `imgx` and `imgy` if needed.

In many cases, the high gradient magnitude region along an edge will be quite thick. For finding lines its best to have edges that are a single pixel wide. Towards this end, make your edge filter implement non-maximum suppression, that is for each pixel look at the two neighboring pixels along the gradient direction and if either of those pixels has a larger gradient magnitude then set the edge magnitude at the center pixel to zero. Map the gradient angle to the closest of 4 cases, where the line is sloped at almost 0°, 45°, 90°, and 135°. For example, 30° would map to 45°.

For more details about non-maximum suppression, please refer to the last page of this handout.

Your code cannot call on Matlab's `edge` function, or any other similar functions. You may use `edge` for comparison and debugging. A sample result is shown in Figure 1.

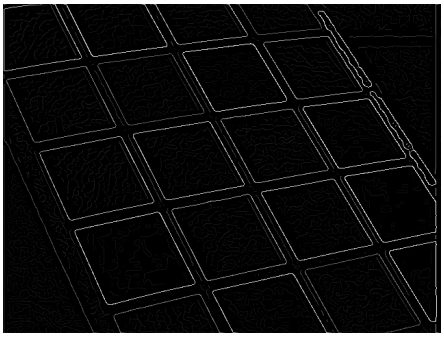


Figure 1: Edge detection result.

Q2.3 The Hough transform (20 points)

Write a function that applies the Hough Transform to an edge magnitude image. Display the output for one of the images in the write-up.

```
function [H, rhoScale, thetaScale] = myHoughTransform(Im, threshold,
                                                      rhoRes,thetaRes)
```

`Im` is the edge magnitude image, `threshold` (scalar) is a edge strength threshold used to ignore pixels with a low edge filter response. `rhoRes` (scalar) and `thetaRes` (scalar) are

the resolution of the Hough transform accumulator along the ρ and θ axes respectively. `H` is the Hough transform accumulator that contains the number of "votes" for all the possible lines passing through the image. `rhoScale` and `thetaScale` are the arrays of ρ and θ values over which `myHoughTransform` generates the Hough transform matrix `H`. For example, if `rhoScale(i)` $= \rho_i$ and `thetaScale(j)` $= \theta_i$, then `H(i,j)` contains the votes for $\rho = \rho_i$ and $\theta = \theta_i$.

First, threshold the edge image. Each pixel (x, y) above the threshold is a possible point on a line and votes in the Hough transform for all the lines it could be a part of. Parametrize lines in terms of θ and ρ such that $\rho = x \cos\theta + y \sin\theta$, $\theta \in [0, 2\pi]$ and $\rho \in [0, M]$. M should be large enough to accommodate all lines that could lie in an image. Each line in the image corresponds to a unique pair (ρ, θ) in this range. Therefore, θ values corresponding to negative ρ values are invalid, and you should not count those votes.

The accumulator resolution needs to be selected carefully. If the resolution is set too low, the estimated line parameters might be inaccurate. If resolution is too high, run time will increase and votes for one line might get split into multiple cells in the array.

Your code cannot call Matlab's `hough` function, or any other similar functions. You may use `hough` for comparison and debugging. A sample visualization of `H` is shown in Figure 2.

Figure 2: Hough transform result.

Q2.4 Finding lines (15 points)

Write a function that uses the Hough transform output to detect lines,

```
function [rhos, thetas] = myHoughLines(H, nLines)
```

where `H` is the Hough transform accumulator, and `nLines` is the number of lines to return. Outputs `rhos` and `thetas` are both `nLines`×1 vectors that contain the row and column coordinates of peaks in `H`, that is, the lines found in the image.

Ideally, you would want this function to return the ρ and θ coordinates for the `nLines` highest scoring cells in the Hough accumulator. But for every cell in the accumulator corresponding to a real line (likely to be a locally maximal value), there will probably be a number of cells in the neighborhood that also scored high but should not be selected. These non maximal neighbors can be removed using non maximal suppression. Note that this non maximal suppression step is different from the one performed earlier. Here you will consider all neighbors of a pixel, not just the pixels lying along the gradient direction. You can either implement your own non maximal suppression code or find a suitable function on the Internet (you must acknowledge and cite the source in your write- up, as well as hand in the source in your `matlab/` directory). Another option is to use Matlab function `imdilate`.

Once you have suppressed the non maximal cells in the Hough accumulator, return the coordinates corresponding to the strongest peaks in the accumulator.

Your code can not call on Matlab's `houghpeaks` function or other similar functions. You may use `houghpeaks` for comparison and debugging.

Q2.5 Fitting line segments for visualization (5 points)

Now you have the parameters ρ and θ for each line in an image. However, this is not enough for visualization. We still need to prune the detected lines into line segments that do not extend beyond the objects they belong to. This is done by `houghlines` and `drawLines.m`. See the script `houghScript.m` for more details. You can modify the parameters of `houghlines` and see how the visualizations change. As shown in Figure 3, the result is not perfect, so do not worry if the performance of your implementation is not good. You can still get full credit as long as your implementation makes sense.

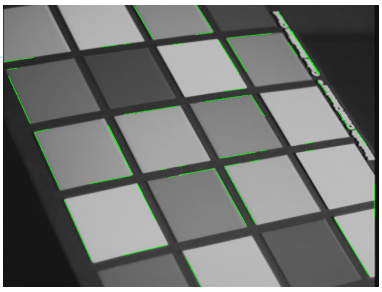


Figure 3: Line segment result.

Q2.5x Implement *houghlines* yourself (extra: 10 points)

In Q2.5, we used the Matlab built-in function `houghlines` to prune the detected lines into line segments that do not extend beyond the objects they belong to. Now, it's our turn to implement one ourselves! Please write a function named `myHoughLineSegments` and then compare your results with the Matlab built-in function in your write-up. Show at least one image for each and briefly describe the differences.

```
function [lines] = myHoughLineSegments(lineRho, lineTheta, Im)
```

Your function should output `lines` as a Matlab array of structures containing the pixel locations of the start and end points of each line segment in the image. The start location of the ith line segment should be stored as a 2×1 vector `lines(i).start` and the end location as a 2×1 vector in `lines(i).stop`. Remember to save your implementation in the `ec/` directory.

Your code can not call on Matlab's `houghlines` function, or any other similar functions. You may use `houghlines` for comparison and debugging.

3 Experiments

Q3.1 (15 points)

Use the script included to run your Hough detector on the image set and generate intermediate output images. Include the set of intermediate outputs for one image in your write-up. Did your code work well on all the image with a single set of parameters? How did the optimal set of parameters vary with images? Which step of the algorithm causes the most problems? Did you find any changes you could make to your code or algorithm that improved performance? In your write-up, you should describe how well your code worked on different images, what effect do the parameters have and any improvements you made to your code to make it work better.

4 Try your own images!

Q4.1x Implement *houghlines* yourself (extra: 10 points)

Take five pictures, either with a camera of your own, or from the Internet. Write a script `ec.m` to take care of reading in your images (use a relative path here, not absolute), making function calls to the various steps of the Hough transform, and generating images showing the output and some of the intermediate steps (like `houghScript.m`). Submit your own images and `ec.m` in `ec/`. Please include resulting images in your write-up.

5 Non-maximum suppression

Non-maximum suppression (NMS) is an algorithm used to find local maxima using the property that the value of a local maximum is greater than its neighbors. To implement the

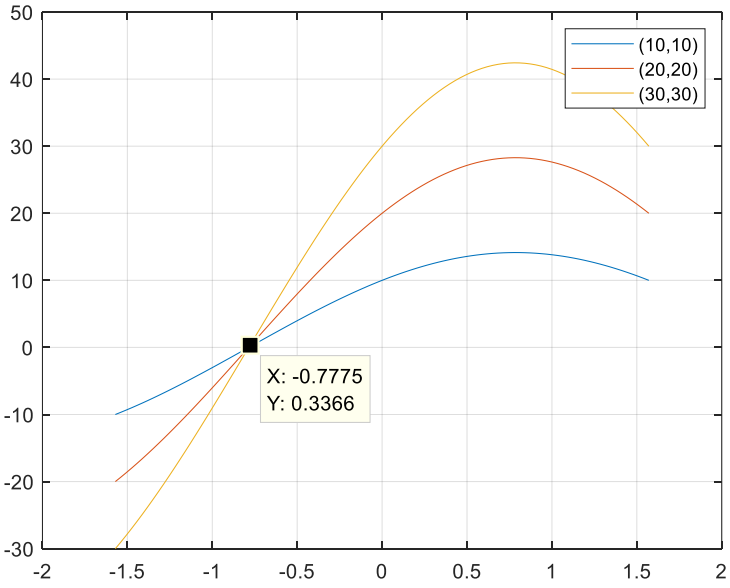
NMS in 2D image, you can move a 3×3 (or 7×7, etc.) filter over the image. At every pixel, the filter suppresses the value of the center pixel (by setting its value to 0) if its value is not greater than the value of the neighbors. To use NMS for edge thinning, you should compare the gradient magnitude of the center pixel with the neighbors along the gradient direction instead of all the neighbors. To simplify the implementation, you can quantize the gradient direction into 8 groups and compare the center pixel with two of the 8 neighbors in the 3×3 window, according to the gradient direction. For example, if the gradient angle of a pixel is $30°$, we compare its gradient magnitude with the north-east and south-west neighbors and suppress its magnitude if it's not greater than these two neighbors.

Credit

This project is adapted directly from Ioannis Gkioulekas.

Q1.1 Hough Transform Line Parametrization

1. $\rho = x\cos\theta + y\sin\theta = \sqrt{x^2+y^2}\left(\frac{x}{\sqrt{x^2+y^2}}\cos\theta + \frac{y}{\sqrt{x^2+y^2}}\sin\theta\right)$
   Let $\frac{x}{\sqrt{x^2+y^2}} = \cos\phi$, then $\frac{-y}{\sqrt{x^2+y^2}} = \sin\phi$
   $\rho = \sqrt{x^2+y^2}(\cos\theta\cos\phi - \sin\theta\sin\phi)$
   $\rho = \sqrt{x^2+y^2}\cos(\theta+\phi), \phi = -\tan^{-1}\frac{y}{x}$
   Amplitude is $\sqrt{x^2+y^2}$
   Phase is $-\tan^{-1}\frac{y}{x}$
2. A line is parameterized in terms of (θ, ρ) as opposed to slope and intercept because it has a lower range of values and allows the image processor to process the lines easily.
   $\frac{\rho}{\sin\theta} = \frac{x}{\tan\theta} + y$
   $y = -\frac{1}{\tan\theta}x + \frac{\rho}{\sin\theta}$
   Slope is $-\frac{1}{\tan\theta}$
   Intercept is $\frac{\rho}{\sin\theta}$
3. Since the graph of line in Hough space is a sinusoid, the maximum ρ is the largest amplitude possible by any point. This is equivalent to $\rho_{max} = \sqrt{W^2+H^2}$. Since we want to plot the full line, we need the complete range of the angle θ. Therefore the range of θ is 0 to 2π
4. $(10,10) \rightarrow \rho = 10\cos\theta + 10\sin\theta$
   $(20,20) \rightarrow \rho = 20\cos\theta + 20\sin\theta$
   $(30,30) \rightarrow \rho = 30\cos\theta + 30\sin\theta$
   The plots are as follows. The intersection is $\theta = -0.7775, \rho = 0.3366$. The slop and intercept of the line are $m = -\frac{1}{tan-0.7775} = 1.0159, b = \frac{0.3366}{\sin 0.7775} = 0.4798$.



Hough space plots

Q2.1 Convolution

The result of the image convolution with a Gaussian filter with sigma 2 is shown below. We can see that the lines are slightly blurred.

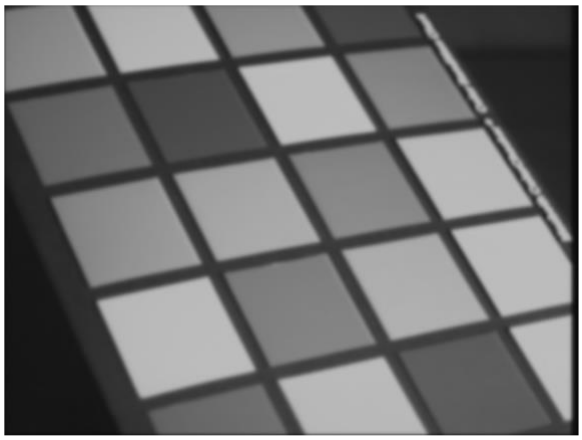


Image result after convolution.

Q2.2 Edge Detection

The result of the edge detection filter is shown below. Using the filter imaged above, we apply Sobel filter to compute the gradients along x and y axes. To further decrease the thickness of edges, we apply non-maximum suppression to all the pixels.

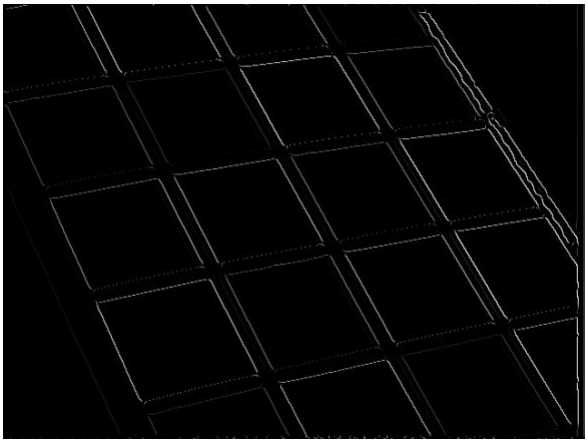


Image result after edge detection filter.

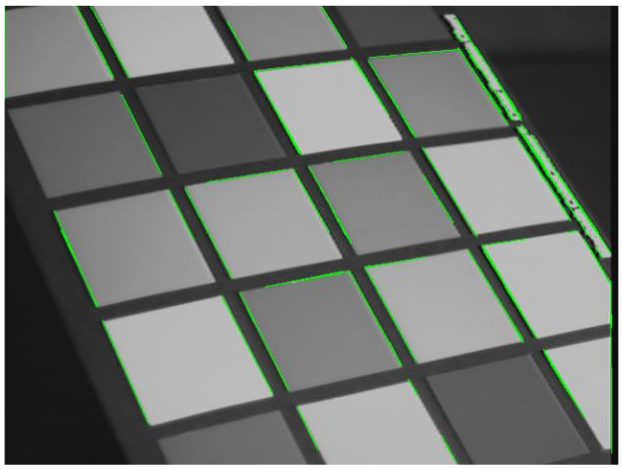
Q2.3 The Hough Transform

The Hough transform result is shown below. We can see sinusoids on the image below. The bright points correspond to the longest lines in the image.

Hough transformation of the edges.

Q2.4 & Q2.5 Finding lines & Fitting line segments for visualization

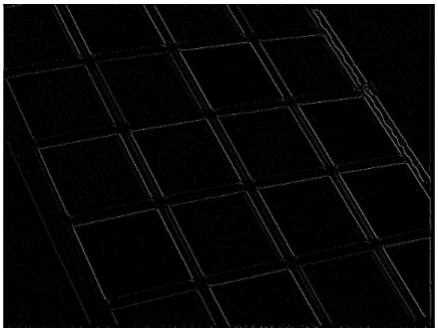
Using the top 50 voted points in the Hough transform. We draw the line segments using the edges as reference. The result is shown below.



Result with highlighted lines.

Q3.1 Experiments

Shown below are the intermediate outputs for an input image. First, it displays the detected edges on each image after it is converted into gray scale. Then, a threshold is applied to extract the lines present on the image. The Hough transformation is applied to the image threshold to complete it. From the top voted points in the Hough space, we draw the lines over the original image.



Edge detection filter result.

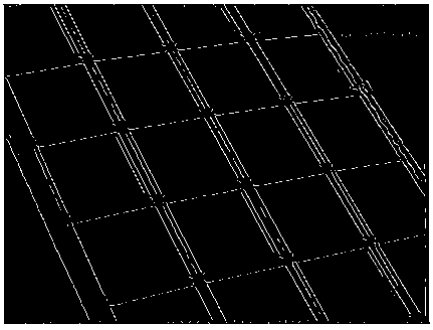
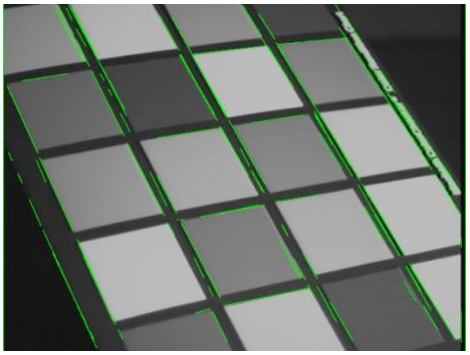


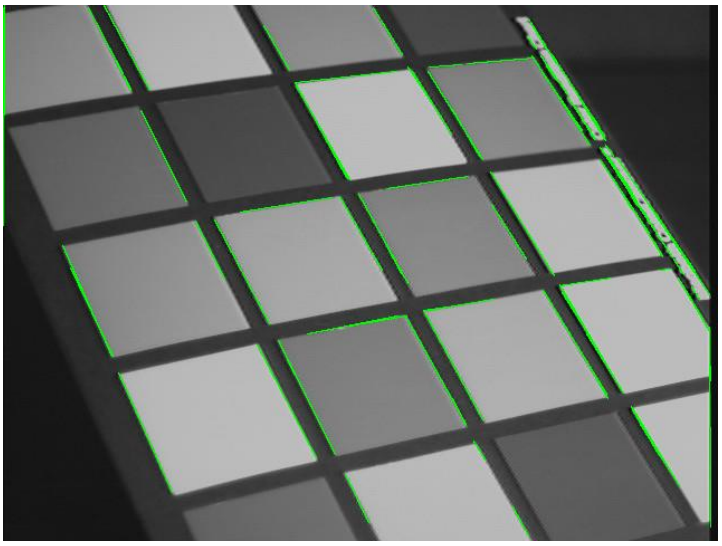
Image result after application of threshold.

Hough space transform of all the points.



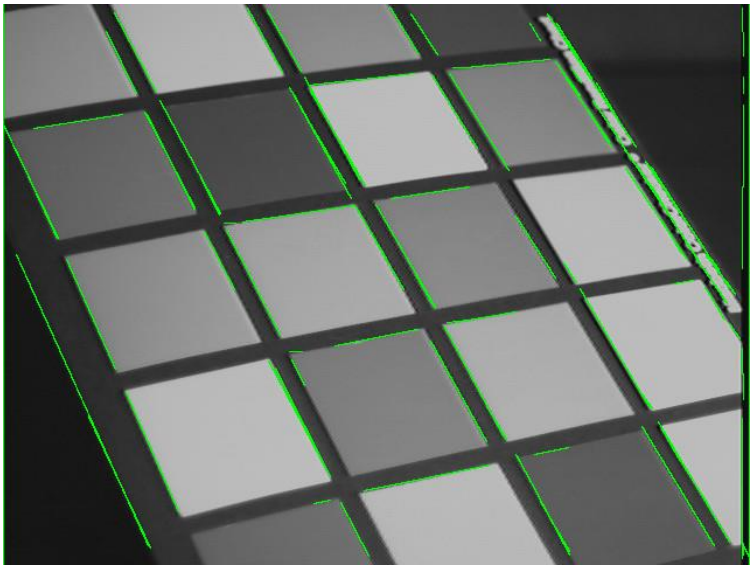
Drawing overlay of Hough lines on the original image.

The hardest part on this project is getting the parameters and algorithm correct for the edge detection filter. As you can see in the image above, there are extra parallel lines to the edges. This is because these extra lines were not eliminated in the edge detection step. One way to mitigate is to increase the threshold to eliminate these. After increasing the threshold to 0.3, we have eliminated a lot of the extra parallel lines. The updated result is shown below.



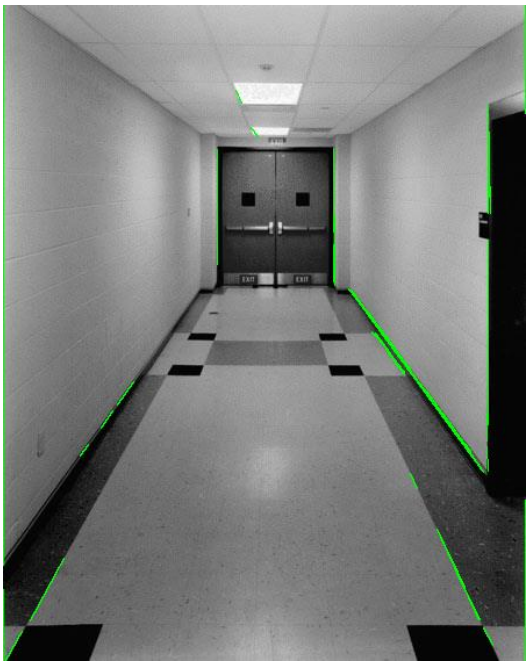
Hough lines with increased threshold.

We can also increase the sigma to spread the edges and makes it easier for the edge detection filter to extract the edges. This will also minimize extra parallel lines present using the original parameters. The result of increasing the sigma is shown below.

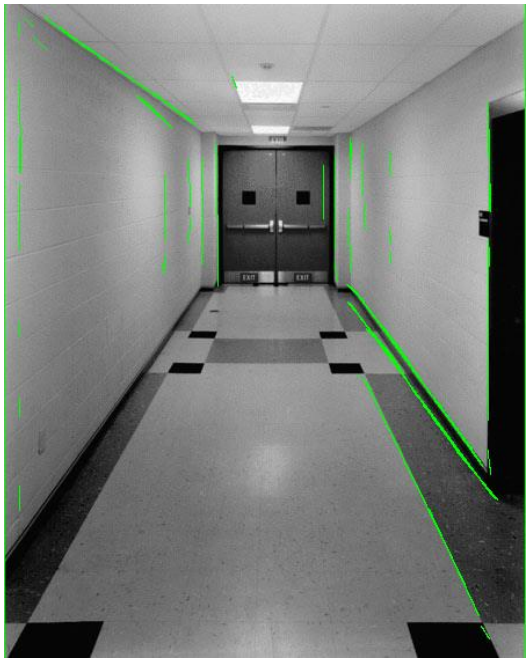


Hough lines with increased sigma.

Comparing threshold and sigma, increasing the sigma has better result because it helps the edge detection filter eliminate the thickness of edges by spreading it further. The spread helps the Non-Maximum Suppression to work better. We can see this on other images.



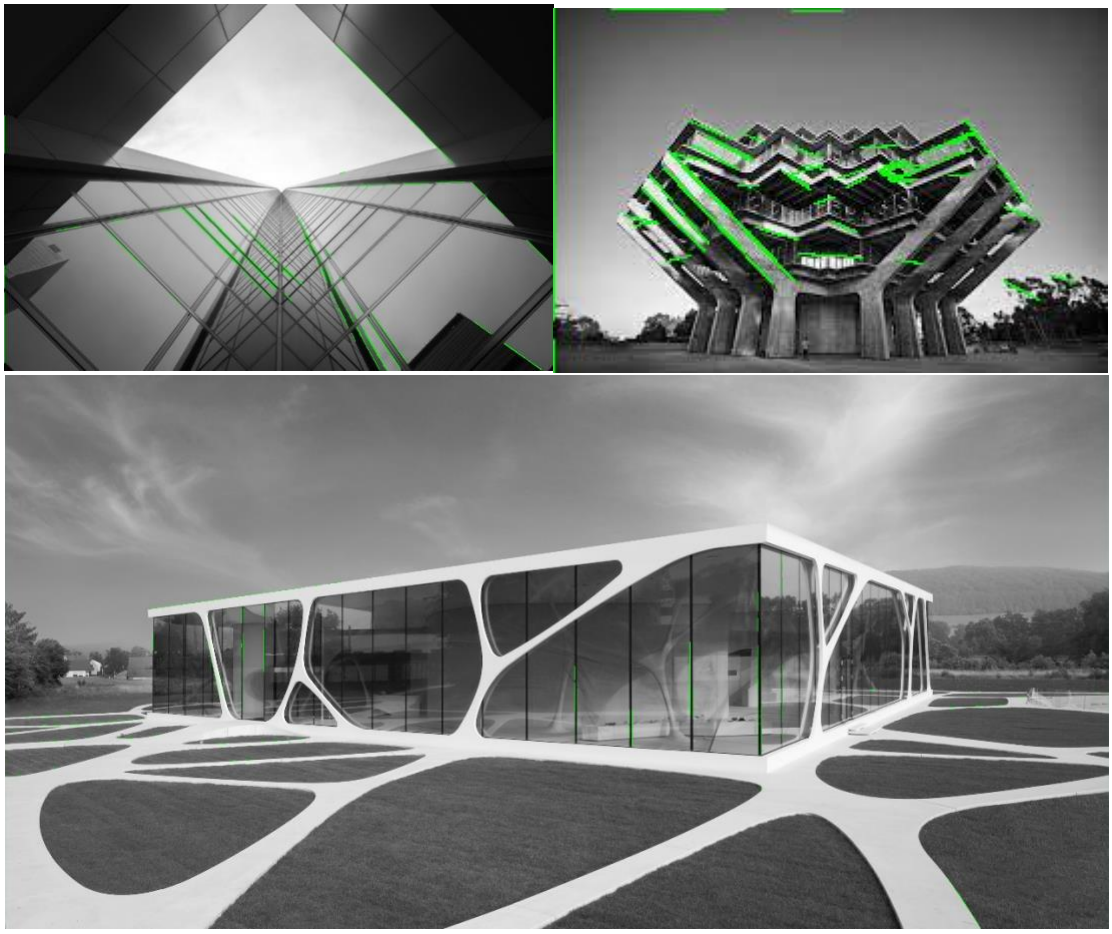
Hough lines with increased threshold.



Hough lines with increased sigma.

Q4

We run it on new 5 pictures to identify the lines on each image. The results are shown in the images below.



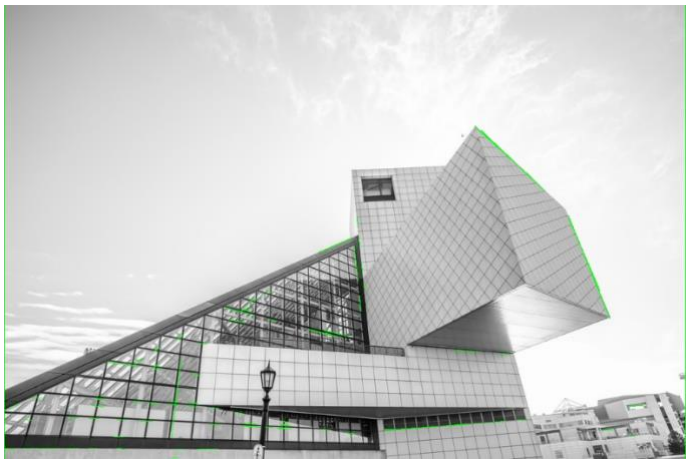


Image results for 5 new pictures