16-385 Assignment Homework 1

Jessica Cheng
jycheng1@andrew.cmu.edu
Collaborated with: Eshani Mishra
February 15, 2017

Q2.1

1.
We can relate x, y and φ with the following equation:

$$\frac{x\cos\theta}{\sqrt{x^2+y^2}} + \frac{y\sin\theta}{\sqrt{x^2+y^2}} = \frac{\rho}{\sqrt{x^2+y^2}}$$

If we have a right triangle with legs x,y and angle across from y of φ, then

$$\cos\phi = \frac{x}{\sqrt{x^2+y^2}}$$

so

$$\cos\phi\cos\theta + \sin\phi\sin\theta = \frac{\rho}{\sqrt{x^2+y^2}}$$

$$\cos(\phi+\theta) = \frac{\rho}{\sqrt{x^2+y^2}} \qquad (1)$$

The amplitude is $\sqrt{x^2+y^2}$ and the phase of the sinusoid is φ. So, equation (1) relates the two.

2. We parametrize the line in terms of ρ and θ because the ranges for m and b are too large. They can both go to infinity, so there are too many choices and therefore too many possible lines. ρ and θ both have limited ranges, so it is more accurate. The slope and intercept can be represented in terms of ρ and θ as:

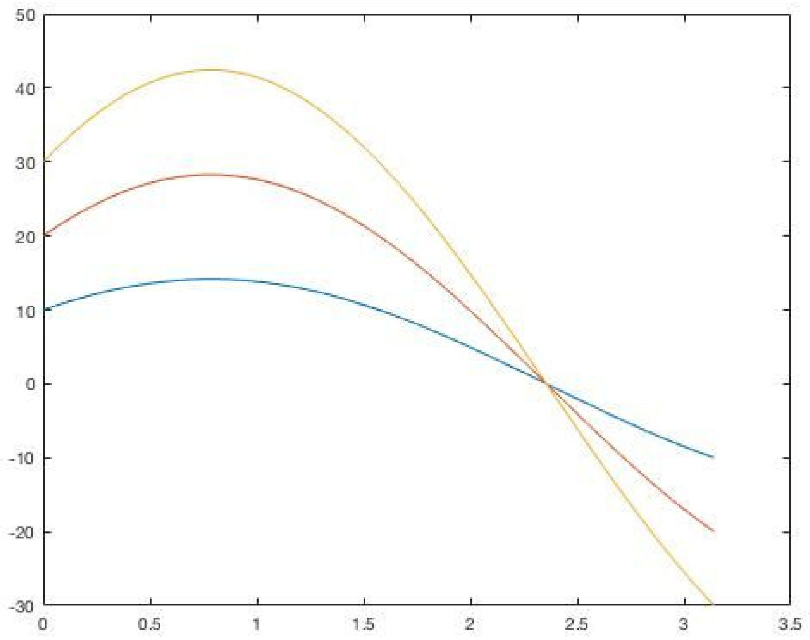
$$\rho = \frac{y-b}{m}\cos\theta + (mx+b)\sin\theta$$

3. With the top left corner as the origin, the maximum absolute value of ρ is the amplitude at (x,y), which is

$$\sqrt{x^2+y^2}$$

The range for θ is 0 to 2*π.

4.



The intersection can be used to find m (slope) and b (y-intercept) through the following equation:
ρ=xcosθ+ysinθ. From our graph, we know that the intersection is at ρ = 0, θ = ¾π .We can rewrite y = mx + b as x = (y-b)/m. Substituting into the original equation, we get:

$$\rho = \frac{y-b}{m}\cos\theta + (mx+b)\sin\theta$$

Evaluating this further with our values of ρ and θ, we get:

$$0 = \frac{y-b}{m}\cos(3\pi/4) + (mx+b)\sin(3\pi/4)$$
$$0 = \frac{y-b}{m}\left(\frac{-1}{\sqrt{2}}\right) + (mx+b)\frac{1}{\sqrt{2}}$$
$$0 = \frac{y-b}{m}(-1) + (mx+b) * 1$$
$$0 = -y + b + m * (mx+b)$$

To find our values of m and b, we can get 2 more equations substituting in (10,10),(20,20) or (30,30) into y and x. Say we use (10,10) and (20,20). The equations to solve for m and b are:

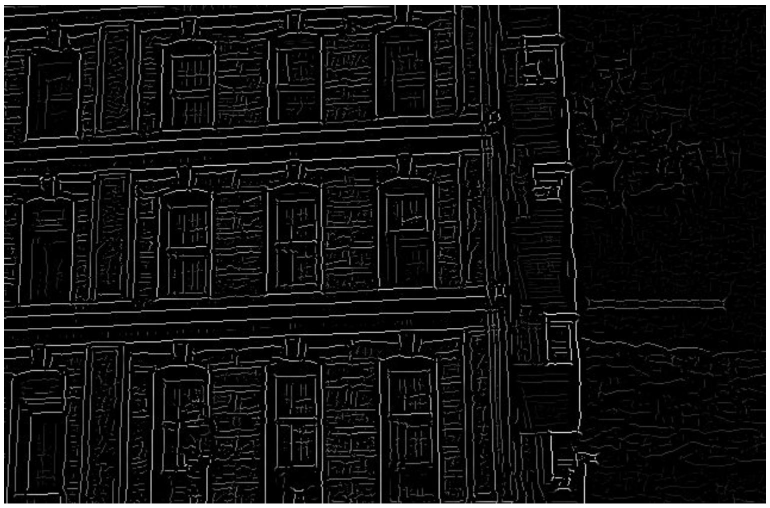
$$0 = -10 + b + m * (m * 10 + b)$$
$$0 = -20 + b + m * (m * 20 + b)$$

We get that m = 1, b = 0.

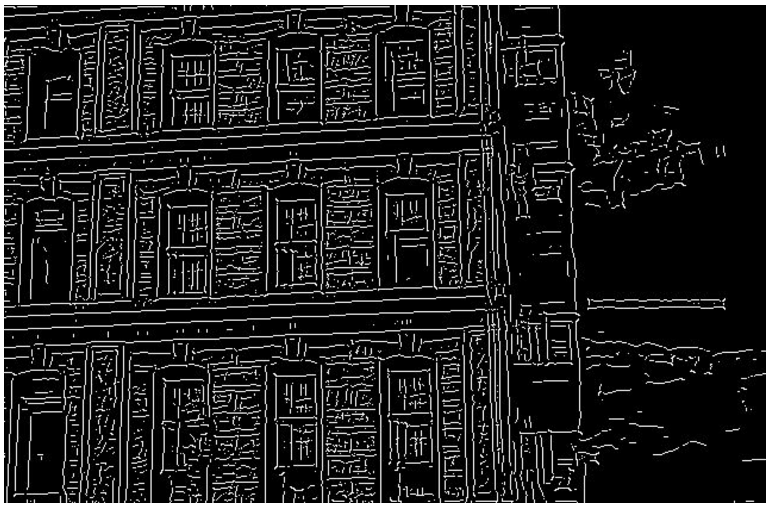
5 Experiments

** All images are from the original parameters **

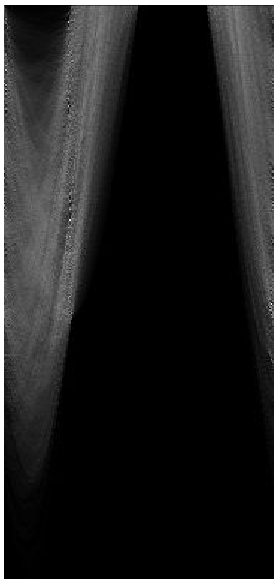
img08_01edge



img08_02threshold

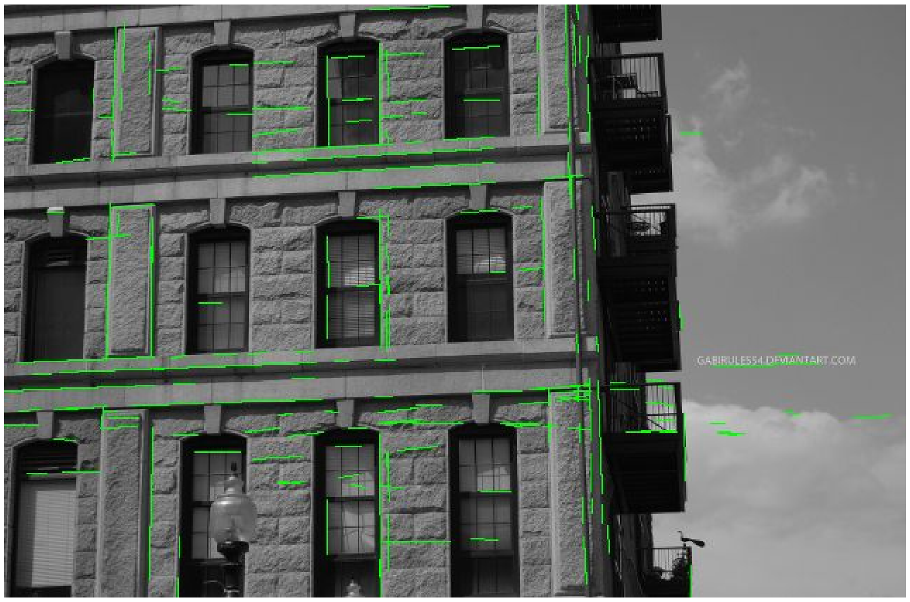


img08_03hough

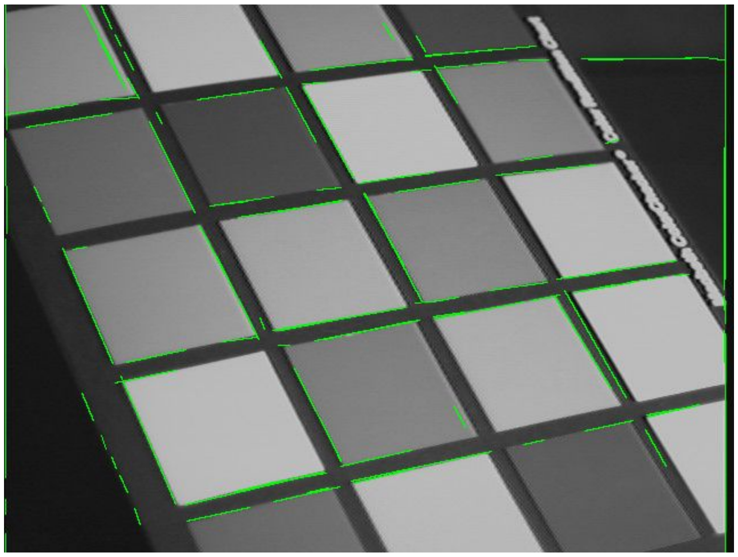


All result images

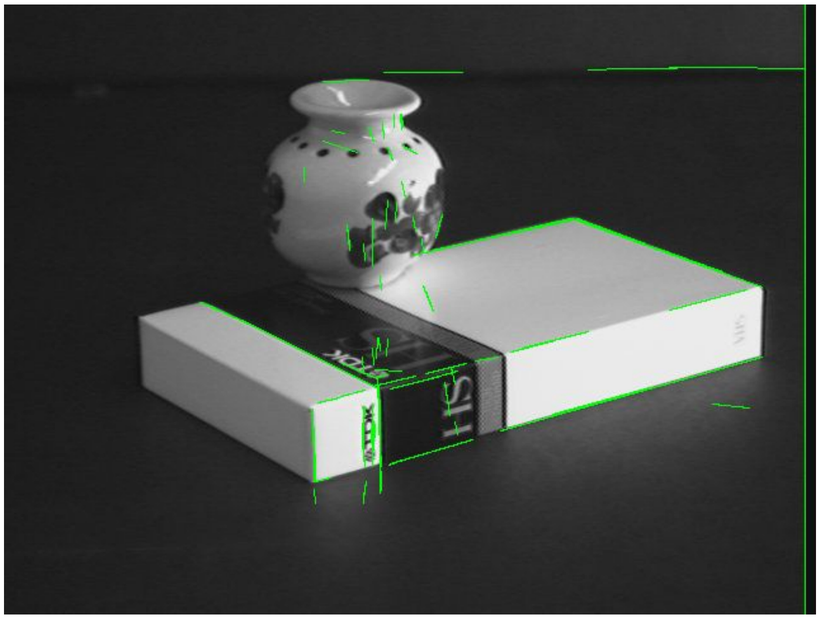
img08_04lines



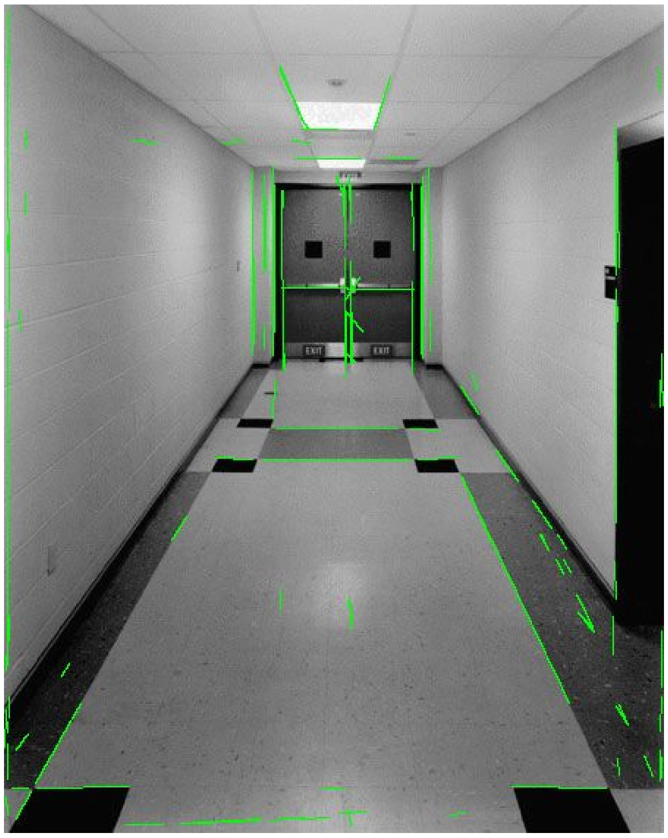
Img01_04lines



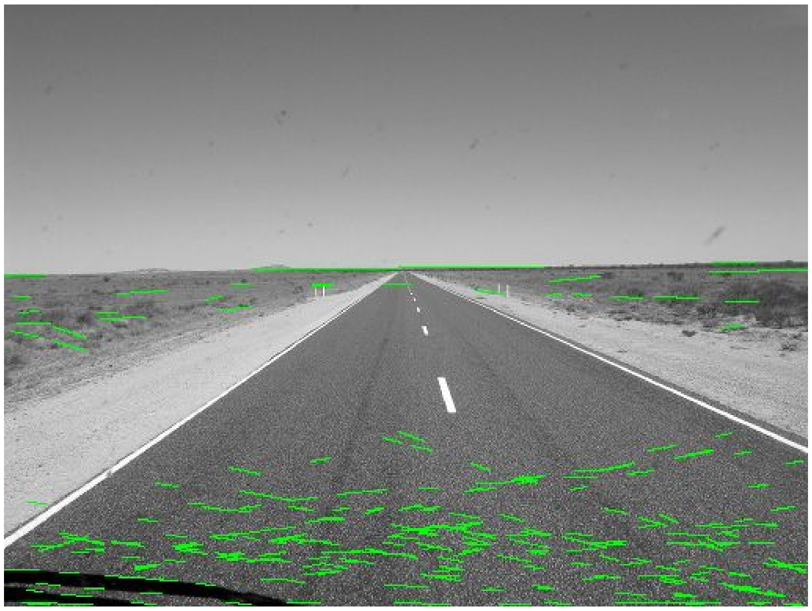
Img02_04lines



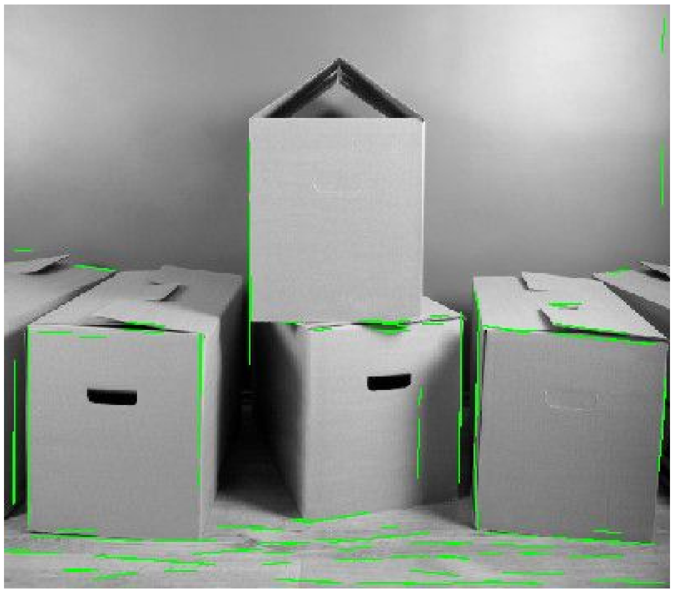
Img03_04lines



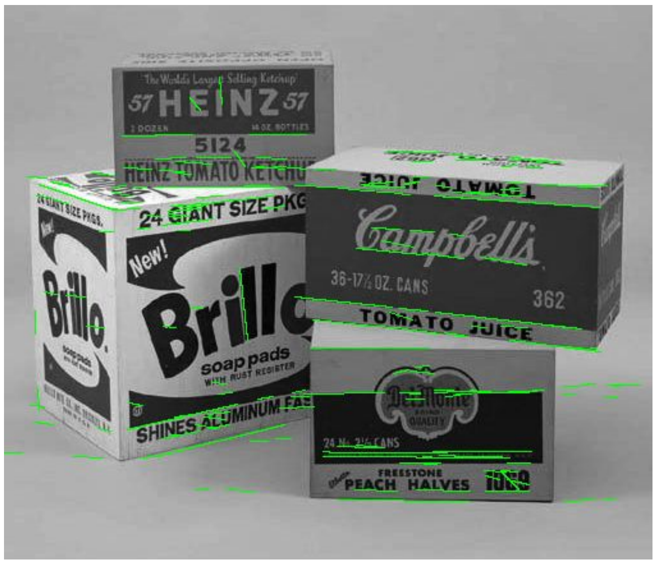
Img04_04lines



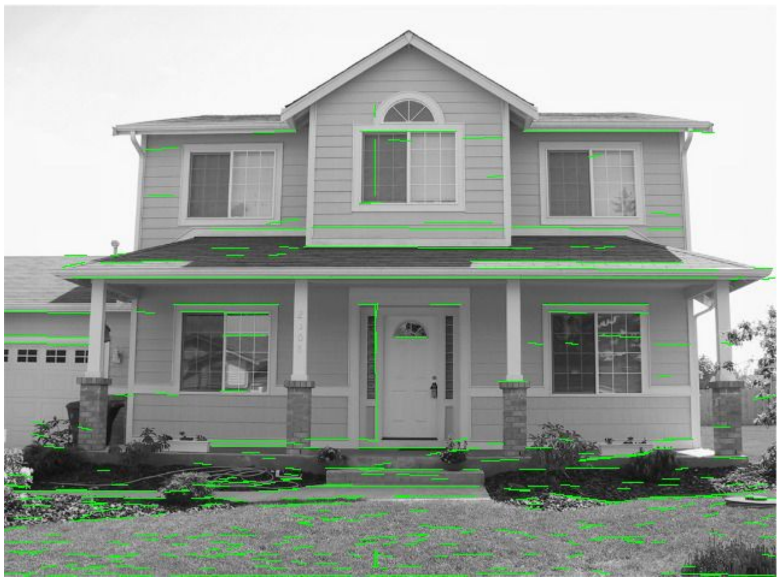
Img05_04lines



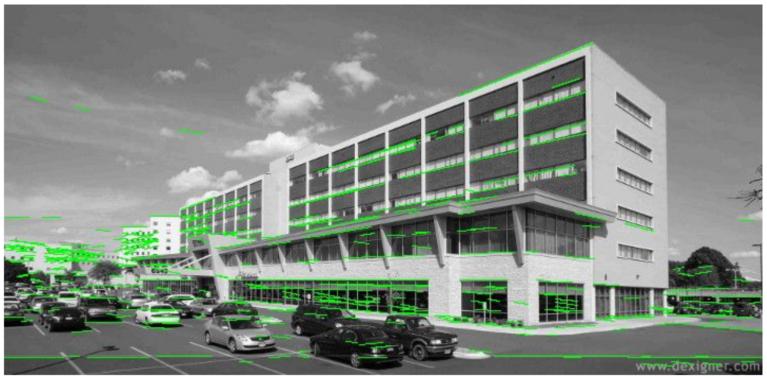
Img06_04lines



Img07_04lines



Img09_04lines



The code worked well on some images and poorly on others (with the original parameters). For the most part, there were more horizontal green lines than vertical ones. Especially in Img04_04lines, the image with the original parameters had a lot of green lines at the bottom. However, upon increasing the parameters, the green lines at the bottom disappeared. However, the original parameters worked really well on Img09_04lines, and changing the threshold yielded worse results. So, because each image has different features, the ideal parameters are different with each image. The two parameters that had the largest effect on the images were threshold and nLines.

Increasing the threshold improved results in Img04_04lines and Img03_04lines. There were fewer stray lines at the bottom when this happened. It also improved Img05_04lines, as the algorithm detected an edge at the back of the boxes, which it missed with the original parameters. It however had adverse effects on Img06_04lines. It actually caused more random lines in top ketchup box. It had relatively little effect on img08_04lines and Img09_04lines.

Increasing nLines resulted in more vertical lines in Img01_04lines, giving a more accurate image. It also put more lines along the floor of Img03_04lines. Increasing nLines put more green lines at the back of the boxes in Img05_04lines, which also improved it. This change is actually less optimal for Img06_04lines, as it put more random green lines across the face of the boxes. While it didn't improve the vertical line detection on img08_04lines, it did pick up more horizontal lines. This change didn't affect the rest of the images in major ways.

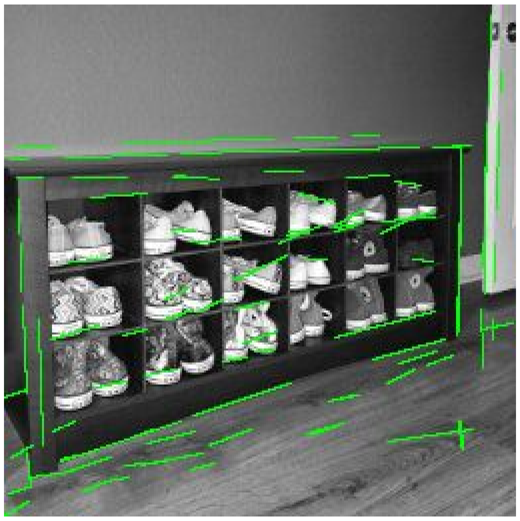
One of the problem-causers is the fact I used sqrt in myEdgeFilter. This actually caused the previously mentioned green lines in Img04_04lines and likely other images with similar random green lines. Also, the threshold parameter was hard to find a middle ground for. What worked better on one image didn't guarantee that it would work equally as well on another image. Since it was hard to find a middle ground, keeping 1 parameter for all images may have caused the most problems

One thing I noticed in terms of speed/accuracy is that the smaller the image, the faster you got results. As there are fewer pixels to check, having a smaller images (like I had in the extra credit portion) really sped up the program. Because of this, by avoiding looping, you could also improve efficiency. By vectorizing whenever you can, you can avoid unnecessary loops.

6 Try Your Own Images!

<u>Resulting images:</u>

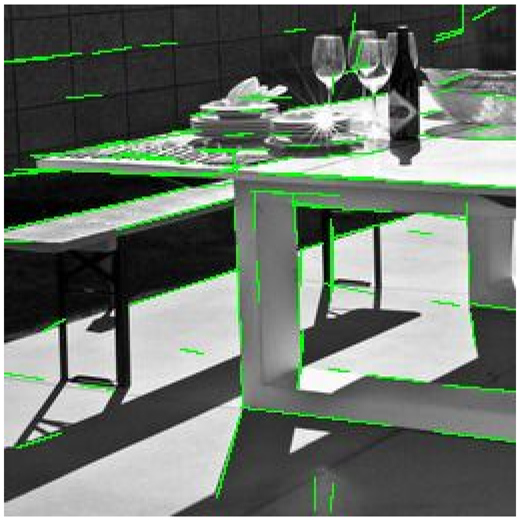
81kJ655XeeL._SL256__04lines



Bshelf_04lines

JxIdTLUR_04lines

Ta_04lines



Tripod_04lines