1 Problem 1

What is the benefit of polar coordinates in the Hough transform?

If the free parameters defining the Hough space axes came from the Cartesian coordinate equation $y = kx + q$, then a very large Hough space image would be required to represent a single point. The set of lines that pass through a point (x, y) includes many very steep ($|k| >> 1$) lines with large intercepts ($|q| >> 1$). As a result, the Hough space image would need to be orders of magnitude larger than the input image, which is often times impractical. Moreover, it would be impossible to represent a vertical line in such a Hough space because it would require $k \to \infty$. Additionally, it would be impossible to represent any lines with $k < 1$ in the discrete Hough space. Thus, choosing the Cartesian parameters would greatly limit the effectiveness of the Hough transform.

The polar formulation that is traditionally used circumvents these limitations because it can parametrize a line by an angle θ with the x axis and its perpendicular distance r to the origin. The polar parametric form of the line allows lines to be represented by finite coefficients in Hough space:

$$\begin{aligned} \theta &: 0 \to 2\pi \\ r &: 0 \to \max(H_{im}, W_{im}) \end{aligned}$$

where H_{im} and W_{im} are the image height and width respectively. Additionally, these variables can be discretized and still retain a good deal of representative power. These features make the polar form a natural coordinate system to employ when searching for lines in images using the Hough transform.

2 Problem 2

Explain why the lines generated by the Hough transform are infinite in length.

The Hough transform specifies a point by the set of lines that pass through it using polar coordinates r and θ. A point (x, y) in real space is thus represented by the curve $\{(r, \theta)_H | \, r = x\cos(\theta) + y\sin(\theta)\}$ in Hough space. A line segment in real space is consequently represented by the intersection of many Hough transform curves corresponding to the different points lying on the line segment.

If the Hough space intersection point, $(r^\star, \theta^\star)_H$ is the only information used to reconstruct the input image, then it corresponds to an infinite line because there is no information encoded in the Hough space point that specifies end conditions for the line segment in real space. One thing to note is that based on the size of the Hough space image and on the values of other points in Hough space, it should be possible to reconstruct a line segment with specific length. It is only when a single point (or small subset of the Hough space) $(r^\star, \theta^\star)_H$ is used that the reconstructed line loses all end point information.

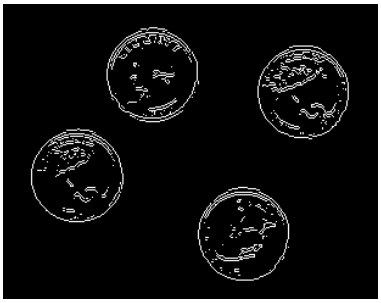
3 Problem 3

Develop and implement a Hough transform that detects circles.

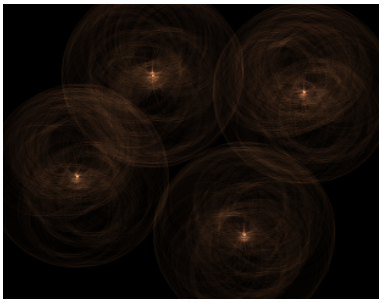
The Hough transform algorithm I implemented here was designed to find circular boundaries of radius 38px, which was measured from the original image in Figure 1a. The coordinates in this Hough space are $(a, b)_H$, which correspond to the only unknown variables (the circle centers) in the equation

$$(x - a)^2 + (y - b)^2 = R^2.$$

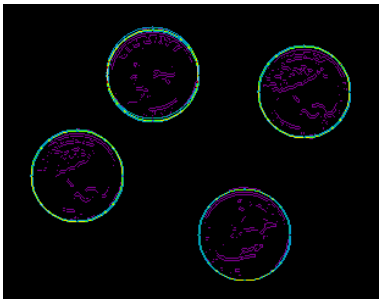
Every point (x, y) in the original image is mapped to the set of points $\{(a, b)_H | \, (x - a)^2 + (y - b)^2 = R^2\}$. The Hough transformed image in Figure 1b is the sum of the transformations of every point in the real space image. The brightest points indicate the Hough space parameters that many real space points could correspond to. Therefore, these bright sections reveal the centers of circles in the original image. I manually



(a) Original Image



(b) Hough Transformed Image



(c) Inverse Hough Transform Overlay

Figure 1: Hough Transformations

thresholded the Hough transformed image and reconstructed circles around the remaining points. The results are displayed in Figure 1c and are overlayed with the original image. The blue and green points are the reconstructed borders. The purple points are the original image. Because the free parameters that defined the Hough space axes were spatial coordinates of the center of circles, the inverse transformation simply required running the thresholded Hough transform image through the transformation algorithm.

problem3.py

```
# Athanasios Athanassiadis Feb 2012
from circhough import *
from scipy.misc import imread
from pylab import cm, imsave

im = imread('img_hough_circle.tif')
ht = hough(im/255.,38)

thresh = 75
im_inv = hough(ht>thresh,38)

imsave('hough_t.png', ht, cmap = cm.copper)
imsave('inverse_hough.png',5*im_inv+im/255, cmap=cm.spectral)
```

4 Appendix: Common Code

The base code used for Problem 3 in this homework is contained in `circhough.py`.

circhough.py

```
# Athanasios Athanassiadis Feb 2012
import numpy as np
from ball import make_ball

def hough(im, R):
    '''
     return Hough transform of a binary image containing circles of radius R
     the axes in the Hough transform represent the a and b center coords
     of a circle given by the equation:

        (x - a)**2 + (y - b)**2 = R**2

```

```
    for each 'on' point, p0, in the binary image, the set of centers of
    circles that could pass through p0 is represented by a circle of
    radius R about p0.  Therefore, for each 'on' point in the image, p0,
    I will add a circle of radius R centered at p0 to the accumulator

    '''
    circ = make_ball(R, shell=1, bgsize=(2*R+1,2*R+1)).astype(int)
    # create accumulator with shape of image since we can only vary the centers
    L,W = im.shape
    acc = np.zeros(im.shape)
    for i in range(L):
        for j in range(W):
            if im[i,j]==1:
                # if near the edges, appropriately crop the circle
                if i<R:
                    i_min = 0
                    crop_top = R - i
                    i_max = i + R
                    crop_bottom = 2 * R
                elif i>L-R:
                    i_min = i - R
                    crop_top = 0
                    i_max = L
                    crop_bottom = L - 1 - i - R
                else:
                    i_min = i - R
                    crop_top = 0
                    i_max = i + R
                    crop_bottom = 2 * R

                if j<R:
                    j_min = 0
                    crop_left = R - j
                    j_max = j + R
                    crop_right = 2 * R
                elif j>W-R:
                    j_min = j - R
                    crop_left = 0
                    j_max = W
                    crop_right = W - 1 - j - R
                else:
                    j_min = j - R
                    crop_left = 0
                    j_max = j + R
                    crop_right = 2 * R

                # add circle centered at i,j to accumulator
                acc[i_min:i_max,j_min:j_max] += circ[crop_top:crop_bottom,
                                                     crop_left:crop_right]

    return acc
```