Problem Set 6: Hough Line and Circle Accumulators
Computational Perception and Artificial Intelligence

Description:

Now we are going to do some computer vision and perception. The Goal is to have the computer use parameterization to identify lines and circles as objects. To do this we will engage in the Hough Accumulator technique. We will write our own implementations of the Hough algorithms. (So no cheating and using the CV2 built in functions for Hough!). For this problem set you will download a zipped folder `ps06.zip` with several preloaded images in the input folder.

Sample Images:

- `sample1.png` : Checkerboard
- `sample2.png` : Noisy Checkerboard
- `sample3.png` : Coins and Vex parts
- `sample4.png` : Hallway
- `sample5.png` : Football Field

You will write one module called `ps06.py` in the standard class structure. This will be a large section of code so be certain to use comments and formatting to make the code readable. Work will be submitted in a zipped folder called `ps06.zip` and emailed to Mr. Michaud. We will do some work in class on this as this problem set should present some 'special' challenges. Good luck!

Setup:

A. Download and unzip the following folder into your Perception Lastname directory

http://www.nebomusic.net/perception/ps06.zip

B. Open the `ps06.py` file with IDLE and place your name in the comments as indicated in the code.

C. Note that you have an output folder in the directory. Several questions in this problem set require you to use the `cv2.imwrite(path, array)` function save the image data to this folder. Make sure the file names are correct.

Part 1: Lines

1. Import `sample1.png` as `img1`. Make a grayscale copy if `img1` and name it `img1_g`.

2. Use the `cv2.Canny()` function and create an edge image from `img1_g` called `img1_edge`. Save `img1_edge` in your output folder as `img1_edge.png`.

3. Write a Hough Accumulator Array function that returns H, a Hough array containing the votes for a parameterized line.

```
def houghLineAccumulator(E):  # E is edge image
    return H
```

Algorithm: Build Hough Array

Given: Edge image E

-Compute d = $2 * \sqrt{(E\ height)^2 + (E\ width)^2}$
-Initialize H(d, 181) = 0
-For each edge point in E(x,y)
  -for theta = -90 to 90 degrees
    d = x*cos(theta) + y*sin(theta)
    H(d, theta) += 1

-Return H

4. Use the `houghLineAccumulator()` function from Question 2 and create a Hough array H from `img1_edge`. Create a normalized image `H_out` from H and save it as `H_out.png` in the output folder. Note the sample code below to normalize and save:

```
# Normalize and Save H
H_out = H.astype(np.float)
H_out = H_out/ H_out.max()
H_out = H_out * 255
H_out = H_out.astype(np.uint8)
cv2.imwrite("output/H_out.png", H_out)
```

5. Write a function `getHoughVectors(H)` that returns a List of vectors [d, theta] from the Hough Array H. You will need to choose a method to threshold H. Try starting with the max of H and work down in value until you get a reasonable number of vectors in return.

```
def getHoughVectors(H):
    return V # List of vectors [d, theta]
```

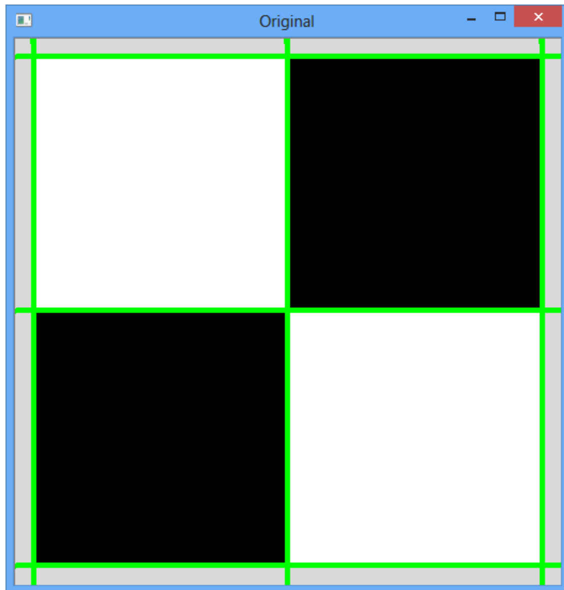
6. Use the `getHoughVectors(H)` function to return a list of vectors from the H from step 3. Using this vector data, use `cv2.rectangle()` and draw white rectangles on the points identified by the vectors. A sample image is shown below.

7. Write a function called `drawLines(img, vectors)` that takes an input of vectors (d, theta) and draws the corresponding lines on the image `img`. Remember that the vectors indicate lines that are 90 degrees (pi/2 Radians) turned from the actual line. You will need to account for this turn when you draw the lines.

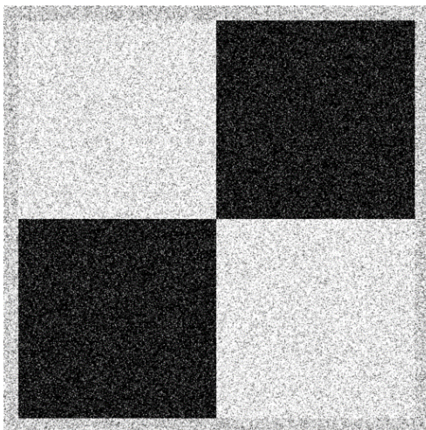
Once you get the point of the line and the angle, choose a distance that will extend beyond the edge of the image. You can use vectors to compute two off image points from the original point and adjusted theta. Once you have two points, use the `cv2.line()` function to draw the line.

```
def drawLines(img, vectors):
    return img # With lines added
```

8. Use the `drawLines()` function from 7 and the `getHoughVectors()` function and draw the lines onto `img1`. Save this image with lines as `img1_lines.png` in the `output` folder. It should look something like: (Green Lines at 3 pixel width)



9. Now that we have a Hough Lines system of functions, let us try it on a noisy image. Import `sample2.png` as `img2`. Make a grayscale copy and call it `img2_g`.



10. Before we make an edge image, we are going to use a Gaussian filter to smooth out the noise. Use the `cv2.GaussianBlur()` function and add some Gaussian noise to `img2_g` and call the new array `img2_noise`. You will need to experiment with the filter size (n x n) and the sigma to get a blur that works. Same `img2_noise` in the output folder as `img2_noise.png`.

11. Make an edge image `img2_edge` from `img2_noise` with the `cv2.Canny()` function.

12. Let's find some lines! Use the `houghLineAccumulator(), getHoughVectors()` and `drawLines()` functions to find and draw the lines on the blurry img2. Save this image as `img2_lines.png` in the output folder.

13. Now we will try line finding with some real world images. Import `sample3.png` as `img3`. Produce a line image from `img3` using the steps of:

-Make GrayScale Image
-Blur Image with Gaussian Filter
-Run the Hough Accumulator

Work to identify lines that outline the long edges of the VEX pieces. Again, tweak the size of the Gaussian Filter, Sigma, and number of lines captured. Save the result as `img3_lines.png` in the output folder.

14. Repeat the process for `sample4.png`. Find work to find the lines in the hallway that intersect at the vanishing point. Save the result as `img4_lines.png`.

Part 2: Circles

15. We will now write a Hough Accumulator for Circles. We will take an input of an edge image E and a radius r to search for circles of a given radius. (This shortens the length of time for a search by concentrating on a single radius).

Algorithm: Hough Circle Accumulator
 	
Given: Edge image E, Radius r
 	
-Initialize array H(height, width) if E with zeros
-For every edge pixel (x,y) in E:
    -For theta 0 to 359
        a = x + r*cos(theta)
        b = y + r*sin(theta)
        H(b, a) += 1
 	
-Return H

Write a function `houghCircleAccumulator(E, r)` that takes inputs of edge image E and radius r and returns a Hough array H with the votes

```
def houghCircleAccumulator(E, r):
    return H
```

16. Write a function `getHoughCircles(H, r)`. This function should return a List of tuples (y, x, r) with x and y being the center of the circle and r being the radius based on voting. Again, you will need to experiment with a threshold to get an appropriate number of votes. This will be very similar in structure to the `getHoughVectors()` function you wrote for question 5.

```
def getHoughCircles(H, r):
    return C  #List of tuples (y, x, r)
```

17. Write a function `drawCircles(img, C)` that takes an input of C (a list of tuples (y, x, r)) and draws a circle for each tuple in C on image I.

```
def drawCircles(img, C):
    return img
```

18. Almost there! Load `sample3.png` into the array `img3`. Use Gaussian blurring and the Canny tools to create an edge image `img3_edge`.

19. Use the `houghCircleAccumulator(), getHoughCircles()`, and `drawCircles()` functions to identify the quarters in the sample3.png. You will need to experiment with the radius and thresholding to get the circles. Save the image with the circles identified as img3_quarters.png in the output folder.

20. Extra Credit 1: Use the `houghCircleAccumulator(), getHoughCircles()`, and `drawCircles()` and identify the nickels and dimes in `sample3.png`. Save the result as `img3_coins.png` in the output folder. (Hint - a great final project idea would be a way to have a computer vision program 'count' money in coins using Hough Accumulators)

21. Extra Credit 2: Import `sample5.png` as `img5`. Tinker with the Hough Lines functions you write until the program only identifies the yard-lines of the football field. (Hint - try identifying and not drawing lines with thetas of 90 (or 0 depending on how you are handling lines) degrees). Save the result as `img5_yardlines.png` in the output folder.

Good Luck!