CSC 489/689 Deep learning in computer vision

Homework 1: simple vision system. (50+20pts)

Posted: Wednesday, September 4, 2019.
Due: 11:59pm, September 18th, 2019

Late policy applies (each late hour 5% reduction after mid-night round up to next hour).

Knowledge required: Python, Numpy, Scipy, Image Convolution, Image gradients, Edge detection.

Required reading: SimpleVision.pdf, Szeliski Chapter 3

Instructions:

1. Finish the python tutorial I uploaded on blackboard last week.
2. Read chapter_01_simplesystem.pdf, Lecture 2 slides.
3. Please type your equation solutions in an document (Latex or word). Please save your file as ps1_yourfirstname_yourlastname
4. Please read course syllabus for collaborative policy.
5. Please submit all of your work in a zipped folder to blackboard. No email. Please do not share your code on Github at least 24 hours after due date.

Problem 0: Basic image processing (10pts).

We wish to set all pixels of in mage that have a value of 10 or less to 0, to remove camera sensor noise. However, our code is slow (below) when run on a database with 1000 grayscale images.

Image: grizzlypeakg.png

```
from skimage import io

A = io.imread('grizzlypeakg.png')
(m1,n1) = A.shape
for i in range(m1):
    for j in range(n1):
        if A[i,j] <= 10 :
            A[i,j] = 0
```

How do you speed it up? Please submit your code. Hint: use pre-allocation and logical indexing.

Problem 1. A simple image formation model (10pts)

The goal of this first exercise is to take images with different settings of a camera to create pictures with perspective projection and with orthographic projection. Both pictures should cover the same piece of the scene. You can take pictures of real places (e.g., the street, a living room, ...) or you can also create your own simple world (e.g., you can print simpleWorld.pdf and create your own scenes. I recommend printing on mate paper).

To create pictures with orthographic projection you can do two things: 1) use the zoom of the Digital camera, 2) crop the central part of a picture. You will have to play with the distance between the camera and the scene, and with the zoom (or amount of cropping) so that both images look as similar as possible only differing in the type of projection (similar to figure 1.4, in the lecture 1 notes). Submit the two pictures and label out clearly which parts of the images reveal their projection types.

Problem 2. Orthographic projection (10pts)

Prove the projection equations (eq. 1.2 and 1.3 in chapter_01_simplesystem.pdf) that relate the coordinates of one point in 3D world and the image coordinates of the projection of the point in the camera plane.

$$x = \alpha X + x_0 \quad (1.2)$$

$$y = \alpha(\cos(\theta)Y - \sin(\theta)Z) + y_0 \quad (1.3)$$

Problem 3. Constraints (10pts)

In the Lecture slide, we have written all the derivative constraints for Y(x,y). Write the constraints for Z(x,y).

Problem 4. Implement edge detection and edge classification algorithm with code. (20pts).

Please submit your code together with your images. Make sure you run your code on virtual box or make sure it runs smoothly with Python 3.6.

Step 1: load your images (of the images you took or the simple world images provided).

Step 2: Extract edges and orientations. (1.4 from chapter)

Here you can use numpy/scipy and a sobel filter. (e.g. scipy.ndimage.sobel).
Or you can convolve your image with your own filter. Here are some tutorials:
https://scipy-lectures.org/advanced/image_processing/auto_examples/plot_find_edges.html

Here are some pseudo code:

```
% Figure/ground separation
ground = double(min(img,[],3)>110);
foreground = double(ground==0);

m = mean(img,3);
dmdx = scipy.signal.convolve2d(m, [-1 0 1; -2 0 2; -1 0 1], 'same');
dmdy = = scipy.signal.convolve2d (m, [-1 0 1; -2 0 2; -1 0 1]', 'same');

% Edge strength (edge magnitude)
mag = math.sqrt(dmdx.^2+dmdy.^2);

% Edge orientation
theta = atan2(dmdx, dmdy);
edges = mag>30;
```

Step 3: Classify edges

```
% Who owns the boundaries? the ground owns no boundaries. We set to zero
% all edges inside the ground.
% First find out the edges in foreground.
edges = edges.*foreground;
% Classify orientation of the edges to vertical and horizontal edges.
```

Hint: using edge orientation?

```
% computing occlusion and contact edges.

% plot the edges on the images. Plot contact edges, occlusion edges, vertical edges and
horizontal edges

% show normal
```

Output figures are similar to Figure 1.8 in chapter_01_simplesystem.pdf

Problem 5 (extra credit, 20pts)

Write the code to solve for Z (recover the depth) using the constraints. Follow the 1.5.5 and 1.5.6 in chapter_01_simplesystem.pdf

Step 1: Initialize the variables. Here are pseudo codes:

```
Nconstraints = nrows*ncols*20;
Aij = zeros([3 3 Nconstraints]);  A matrix;
```

ii = zeros([Nconstraints 1]);
jj = zeros([Nconstraints 1]);
b = zeros([Nconstraints 1]);

Hints: for every pixel:

1. Check if current neighborhood touches an edge.
2. Check if current neighborhood touches ground pixels
3. Check if current neighborhood touches vertical pixels
4. Check if current neighborhood touches horizontal pixels
5. Orientation of edge (average over edge pixels in current neighborhood).

Step 2: Create linear constraints for Y.
Step 3: Solve for Z based on Y.
% Create sparse matrices

% Solve linear system

Y = A\b;
% Recover 3D world coordinates. Solving X, Z, Y.
e.g. X = x;
Y = -Y
What about Z? Use equtions 1.2, 1.3
Step 4: Render the images.