Intro to Image Understanding (CSC420)
Assignment 4

Posted: March 14, 2018 **Submission Deadline : March 23, 11.59pm, 2018**

Max points: 12 (+ 3 extra credit)

1. In this exercise you are given stereo pairs of images from the autonomous driving dataset KITTI (`http://www.cvlibs.net/datasets/kitti/index.php`). The images have been recorded with a car driving on the road. Your goal is to create a simple system that analyzes the road ahead of the driving car. Include your code to your solution document.

   - In this assignment the stereo results have been provided with the following stereo code: `http://ttic.uchicago.edu/~dmcallester/SPS/spsstereo.zip`. For those interested in actually running the code yourselves (e.g., students doing project 2), please grab it from this link. Since the code doesn't easily install, you are given a few fixed files in CDF_COMPILE.ZIP. Once you download the stereo code, please unzip CDF_COMPILE.ZIP inside the stereo code folder. Then follow the readme instructions provided by the code. Should be easy to install. If not, complain on piazza.
   - We provide detections using the detector Deformable Part Model. You can find the code of this detector in the folder CODE/DPM (the original code can be found here: `http://www.cs.berkeley.edu/~rbg/latent/voc-release5.tgz`). *For those students doing projects requiring detection:* You will need to compile the code, by running COMPILE in Matlab inside the CODE/DPM directory. Make sure you add all the subdirectories to your Matlab path. If you encounter errors during compilation, then commenting out the lines: FV_COMPILE(OPT, VERB); and CASCADE_COMPILE(OPT, VERB); in COMPILE.M should make it to work.
     Note to Python users: DPM has been one of the most used detectors in Computer Vision. Unfortunately, the authors released the code in Matlab. But you can use their code to compute detections in Matlab, and then store them in a Python-friendly format, and do the rest of the assignment in Python.
     Those interested in more recent techniques, you can look at the Faster R-CNN method and code: `https://github.com/rbgirshick/py-faster-rcnn`
   - In the assignment's code folder you will find a few useful functions. In order to use them, please first open GLOBALS.M and correctly set the paths of your data.
   - You are also given a function called GETDATA (Matlab) which will help you to easily browse all the provided data. Look at the function to see all the options it has. It provides you with loading as well as some plotting functionality.
   - In your data directory you are given a folder called TRAIN and TEST. You will need to compute all your results only for the **test** folder. Using TRAIN will be optional. For all but the extra credit exercise, you only need to process **the first three images** written in the DATA/TEST/TEST.TXT file. The easiest to get the list of train/test images is via GETDATA: DATA = GETDATA([], "TEST", "LIST"); IDS = DATA.IDS(1:3);.
   - Matlab: Before running any code, position yourself in the CODE folder and type ADDPATH(GENPATH(PWD)). This will add all the paths to the code you need.

(a) **[2 points]** Describe (with words, no code) how to compute disparity for a pair of parallel stereo cameras. How can you compute depth for each pixel?

(b) **[2 points]** Describe (with words, no code) how to compute the fundamental matrix from a pair of uncalibrated (non-parallel) cameras. Once one has the fundamental matrix, how can one use it for the stereo problem?

(c) **[1 point]** The folder RESULTS under TEST contains stereo results from the algorithm called SPSSTEREO. For each image there is a file with extension _LEFT_DISPARITY.PNG that contains the estimated disparity.
For each image compute depth. In particular, compute DEPTH which is a $n \times m$ matrix, where n is the height and m the width of the original image. The value DEPTH(I,J) should be the depth of the pixel (i, j). In your solution document, include a visualization of the DEPTH matrices. In order to compute depth you'll need the camera parameters:
Matlab users: You'll find the camera parameters via the getData function (pass the "calib" option).
Python users: In the TEST/CALIB folder there are text files with extension _ALLCALIB.TXT which contain all camera parameters
Note that the baseline is given in meters.

(d) **[0 points]** The folder RESULTS under TEST contains detections in a file DETS-TEST.ZIP, which has detection results for each image. The variable DS contains a detection in each row of the matrix. Each row represents a rectangle (called a *bounding box*) around what the detector thinks it's an object. The bounding box is represented with two corner points, the top left corner (x_{left}, y_{top}) and the bottom right corner (x_{right}, y_{right}). Each row of DS has the following information: $[x_{left}, y_{top}, x_{right}, y_{bottom}, id, score]$. Here SCORE is the strength of the detection, i.e., it reflects how much a detector believes there is an object in that location. The higher the better. The variable ID reflects the viewpoint of the detection and you can ignore it for this assignment.

For those of you doing a project that requires detection: I suggest trying to run the detector yourself, since you'll need it for the project. For all test images run the detector for CAR. How to run a detector on one image for car is shown in a function DEMO_CAR. Store the detections (variable DS) in the RESULTS folder. Storing is important as you will need them later and it takes time to compute. Once you compute everything, I suggest that you add a subroutine in getData that loads the detections for each image. Make sure you store also to which class (car, person or cyclist) each detection belongs to.

(e) **[2 points]** In your solution document, include a visualization of the first three images with all the car, person and cyclist detections. Mark the car detections with red, person with blue and cyclist with cyan rectangles. Inside each rectangle (preferably the top left corner) also write the label, be it a car, person or cyclist. (Matlab: Help yourself with the function TEXT.)

(f) **[3 points]** Compute the 3D location of each detected object. How will you do that? Come up with a simple approach to get a (2D) segmentation of each object. Can you use depth to help you?

2. **[2 points]** Explain how Hough voting works for finding circles of known radius r in an image. What if the radius is not known?

3. **Competition in car segmentation [extra credit, up to 3 points]** The goal is to generate a segmentation of all cars in **all 20 test images** as accurately as possible. Each test image should have a value 1 for all pixels that your algorithm believes are car and 0 for the rest of the image. The accuracy is evaluated with the standard intersection-over-union measure. That is, we will compute the number of pixels that you think are car and are actually car in the ground-truth

(this is the intersection part), and all pixels which you as well as ground-truth think are car (this part is union). You can look at and use the evalSeg function for evaluation. You are provided with ground-truth segmentation for all **left train** images. You can look at it with e.g., data = getData("000019", "train", "gt-left-plot");.

Write your ideas and algorithm clearly, and include code. Please also include the folder test/results-seg in your solution zip file. We will evaluate everyone's performance. Extra credit will be given to either very innovative solutions or relative how well you do in the competition. Points will be given as soon as you are at least a little better than your result in exercise 1(e).

Some possible hints: you can for example decrease the threshold of the detector to get more detections for each image. Then you can try to find a threshold on train that gives you the highest segmentation accuracy, an use it in test. To make your segmentations are more smooth you could make use of super pixels. These are computed as a side results of spsstereo. Once you run that code (getDisparity), then you can load super pixels with e.g.: data = getData("000019", 'train', 'superpixels');. You can also use any sophisticated Machine Learning technique you have learned so far. If you are the Neural Network guy/girl, go ahead and use that as well. If you don't have the resources to try something, then still write down your ideas. The goal of this exercise is to get you thinking about the recognition problem!

Best results will be posted on the class webpage.