Intro to Image Understanding (CSC420)
Assignment 3

Submission Deadline : Nov 13 (Thursday), 11.59pm, 2014

Max points: 10, max extra credit points: 5

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 and describes the scene to the driver via text. Include your code to your solution document.

   - You will first need to download and install the following stereo code: `http://ttic.uchicago.edu/~dmcallester/SPS/spsstereo.zip`. Since the code doesn't easily install on CDF, 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.
   - You will need to download the code for the Deformable Part-based Model (DPM) detector: `http://www.cs.berkeley.edu/~rbg/latent/voc-release5.tgz`. You will also need to compile the code, by running COMPILE in Matlab inside the directory where you unzipped the code. 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.
   - 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.
   - One of the useful functions is called GETDISPARITY which should help you compute the disparity for a given stereo pair of images.
   - You are also given a function called GETDATA 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);.

   (a) **[1 point]** Compute the disparity for the test images (remember, only first 3) and store it in the directory called TEST/RESULTS. This is easy via the getDisparity function. For each image then 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 order to compute depth you'll need the camera parameters. You'll find these via the getData function (pass the "calib" option). Note that the BASELINE is given in meters. The best way to store depth for each image pair is probably using .MAT files. In your solution document, include a visualization of the DEPTH matrices.

(b) **[2 points]** For all test images run the detectors for CAR and PERSON and CYCLIST. 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.

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.

(c) **[1 point]** 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. Help yourself with the function TEXT.

(d) **[2 points]** Compute the 3D location of each detected object. How will you do that? Store your results as you'll need them later. *Hint: Use depth information inside each detection's bounding box.*

(e) **[2 points]** We will now perform a simple segmentation of each detected object. By segmentation, we mean trying to find all pixels inside each detection's bounding box that belong to the object. You can do this easily as follows: for each detection you have computed the 3D location of the object. Find all pixels inside each bonding box that are at most 3 meters away from the computed 3D location. Create a segmentation image. This image (a matrix) has the same number of columns and rows as the original RGB image. Initialize the matrix with all zeros. Then for i-th row in DS (which is the i-th detection), assign a value i to all pixels that you found to belong to detection i. In your solution document, include a visualization of the segmentation matrix (for the first three test images).

(f) **[2 points]** Create a textual description of the scene. Try to make it as informative as possible. Convey more important facts before the less important ones. Think what you, as a driver, would like to know first. Think of the camera center as the driver. In your description, generate at least a sentence about how many cars, how many people and cyclists are in the scene. Generate also a sentence that tells the driver which object is the closest to him and where it is. For example, let's say you have an object with location (X, Y, Z) in 3D and LABEL=CAR and you want to generate a sentence about where it is. You could do something like:

```
D = NORM([X,Y,Z]); % THIS IS THE DISTANCE OF OBJECT TO DRIVER
IF X>=0, TXT = "TO YOUR RIGHT"; ELSE TXT = "TO YOUR LEFT"; END;
FPRINTF("THERE IS A %S %0.1F METERS %S\N", LABEL, X, TXT);
FPRINTF("IT IS %0.1 METERS AWAY FROM YOU\N", D);
```

Even if you didn't solve the tasks in (a)-(e), you can still get points for this exercise by writing some pseudo code like the one above. Clearly explain what the piece of code is supposed to do.

2. **Competition in car segmentation [extra credit, up to 5 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 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 techniques 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!

As always, the best results will be posted on the class webpage. We will also post the best descriptions for the exercise 1(f), so use your imagination! A lot of the stuff you do and learn here you'll be able to use when doing the project.