Intro to Image Understanding (CSC420)

Assignment 4

Posted Nov. 8, 2016; Submission Deadline: Nov. 18th 11.59pm 2016

Instructions for submission: Please write a document and submit a **PDF** with your solutions (include pictures where needed). Include your code inside the document, and submit through **MarkUS**.

For full marks you must show your work, not just your final answer.

Max points: 13, max extra credit points: 1.5

1. [**4.5 points**] In this excerise you will map out how to build a vision system. You want to build a robot to take the place of the 'ball boy/girl" during a tennis match. Your robot is modelled after WALL-E (link). Write psudeo-code calling algorithms covered in class to build the robot's visual system and fulfill its destiny. Your robot already knows the following commands to aid you: **faceDirection(X,Y,Z)**; **moveToLocation(X,Y,Z)**; **grabObjectAtLocation(X,Y,Z)**; **victoryDanceAtLocation(X,Y,Z)**. It also comes with camera calibration matrices $K, [R|t]$. Check out a demo Video of a simpler model; note your robot should work during a tennis match, instead of these controlled conditions, but is also equipped with stereo cameras!

   (a) [**1 point**] Brainstorm about the robot's world, what it will encounter, what data it will need for training, what challenges it will face, etc. Create a brainstorming diagram to include in your PDF.

   (b) [**1 point**] Create a flow chart linking together the main processing pipeline the robot will use; reference algorithms from the course (e.g. **cannyEdgeDetection)** and the actions taken, e.g. **moveToLocation(X,Y,Z)**.

   (c) [**2.5 points**] Write psuedo-code to implement your robot, run algorithms from the course by name, e.g. **cannyEdgeDetection** and include relevant arguments. You will be marked on completeness and ingenuity (i.e. how we think your robot will work, the conditions it can handle, etc.).

2. [**8.5 points**] 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.

- 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, please grab it from this link. Since the code doesn't easily install on CDF, you are given a few fixed files in **cdf_compile.zip**. Once you've download the stereo code, please unzip **cdf_compile.zip** inside the stereo code folder. Then follow the readme instructions provided by the code.
- You are provided code for the object detector called Deformable Part Model in the folder code/dpm (the original code can be found here: http://www.cs.berkeley.edu/rbg/latent/voc-release5.tgz). 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, DPM is a powerful object detector and could be trained for other tasks (i.e. projects).

  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.
- 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. 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) **[1 points]** 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 **_ALL CALIB.txt** which contain all camera parameters.

Note that the baseline is given in meters.

(b) **[2 points]** For all test images run the DPM 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 is 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. (Matlab: Help yourself with the function text.)

(d) **[1 point]** Compute the 3D location (centre of mass) 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 bounding 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) **[1.5 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 centre 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 $X \geq 0$, 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 the rest of Q2, 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.

3. **Extra: [1.5 points] total.** (this is an optional exercise) Show (0.75 points) and explain (0.75 points) some interesting initial findings and/or intermediate results from **your component** of your final project. You can use synthetic data, or manual inputs to replace missing modules from your project-mates. **Note: If your project is the autonomous driving project, you must go beyond any code/activity performed for this assignment to be elgibile for the extra credit.**