Intro to Image Understanding (CSC420)

Assignment 4

Posted Oct 31 2018; Submission Deadline: Nov. 7 10:59pm 2018

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: 11.5, max extra credit points: 1

1. [**3.5 points + 1 point extra**] In this exercise you will map out how to build a vision system. You want to build a robot equipped with stereo cameras 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)**, **grabObject(X,Y,Z)** and **victoryDanceAtLocation(X,Y,Z)**. It also comes with camera calibration matrices K, $[R|t]$.
(R: camera rotation; t: move)
Check out a demo Video of a simpler model; note your robot should work during a tennis match, instead of these controlled conditions.
(since during the 'TENNIS MATCH', the condition is out of control -> we need consider avoid matcher & ball)

(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, what its world (model) will look like, 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. For simplicity, your robot should return always return to a default location and do its victory dance there after grabbing a ball (It cannot dance on the field!). Make sure to describe what each function does in your framework. Note that you can add functions in your flowchart if you find that the ones given are not sufficient to do a full cycle.

(c) [**1.5 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.).

(d) **Extra** [**1 point**] Your robot suddenly loses one of its eyes (lens)! Nevertheless, your robot should still be able to achieve its task. Describe in details what you would need to modify (software only) and why those changes are important to solve this issue. Finally, describe in what conditions the robot may still be unable to complete its task after losing an eye. (stereo with structured light)

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. You do not need to run this code (the

output of this code is already provided with the assignment). For those interested in actually running the code yourselves, you may download it from the link above.

- To do object detection, we propose using a simple object detector trained on the MS-COCO dataset with Tensorflow. Follow the tutorial here to install Tensorflow along with the model: Daniel Stang's tutorial (Make sure to move the *models* folder in your *tensorflow* folder). To use the object detector, you can run the Jupyter notebook **object_detection_tutorial.ipynb** mentioned in the tutorial. You will need to add images to the *test_images* folder as mentioned in the tutorial and save *detection_classes*, *detection_scores* and *detection_boxes* from the object *ouput_dict* to do the rest of the assignment (in python or MATLAB).
  Note: for those who would prefer using a different object detector, you are free to do so and/or train your own. You will need to mention which you have used in your report. You do not need to include the detector in your submission.
- 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. You only need to process the **first three images** written in the **data/test/test.txt** file.
- Only submit your code for the following questions.

(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:

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) [**1.5 points**] For all test images run the object detector and keep the results for car, person, bicycle and traffic light. Store the detections in the **results** folder. Storing is important as you will need them later and it takes time to compute.

The rows in array *detection_boxes* 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_{bottom}). The values are by default normalized by the size of the image. Each row has the following information: [x_{left}, y_{top}, x_{right}, y_{bottom}]. The object *detection_scores* reflects the how much the detector believes that there is a specific *detection_classes* object inside the bounding box. In this assignment, you only need to use the following classes: 1 (person), 2 (bicycle) ,3 (car) and 10 (traffic_light). You will need to use a threshold to only keep strong bounding box proposals. Describe what threshold you have used in your method.

(c) [**1 point**] In your solution document, include a visualization of the first three images with all the car, person, bicycle and traffic light detections you have kept. Mark the car detections with red, person with blue, cyclist with green and traffic light with cyan rectangles. Inside each rectangle (preferably the top left corner) also write the label, be it a car, person, cyclist or traffic light. The visualization should look similar to the one shown in the notebook.

(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 the *i*-th detection (bounding box), 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 image (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 and if there is a traffic light nearby. 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 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.