University of Toronto

AER-1515 – Assignment3

Tushar Aggarwal

Student Number: 999356913

12-14-2020

Name: Tushar Aggarwal Std# 999356913

AER-1515 ASSIGNMENT 3

The provided code was imported into a Jupyter notebook for the ease of programming. The notebook provides headers for each part of the assignment. The code was first trained on the training images with ground truth and then implemented on the test set (for which the ground truth was not available).

The code for the training section is available in the notebook, however, it is commented out to avoid any confusion.

PART 1 – DEPTH ESTIMATION

Part one of the assignment asked us to do depth estimation on the given disparity images along with camera calibration information.

This part was implemented in the following manner:

1. Disparity images were loaded into the Jupyter notebook using OpenCV library
2. Camera calibration were loaded into the Jupyter notebook using file library
3. StereoCalib class was used to read the camera calibration for camera 2 (left) and 3 (right)
4. From the StereoCalib class focus and baseline information was extracted
5. Using the extruded information in point 4 depth was calculated using the following formula:

$$Depth = \frac{Focus * Baseline}{Disparity}$$

   a. Disparity values with 0 were not used for calculating and were left as zero
6. Depth values less than 10cm (0.1) and more than 80m were set to '0' in the depth maps.

The following are the depth maps for test images:

Image: 000011

Image: 000012

Image: 000013

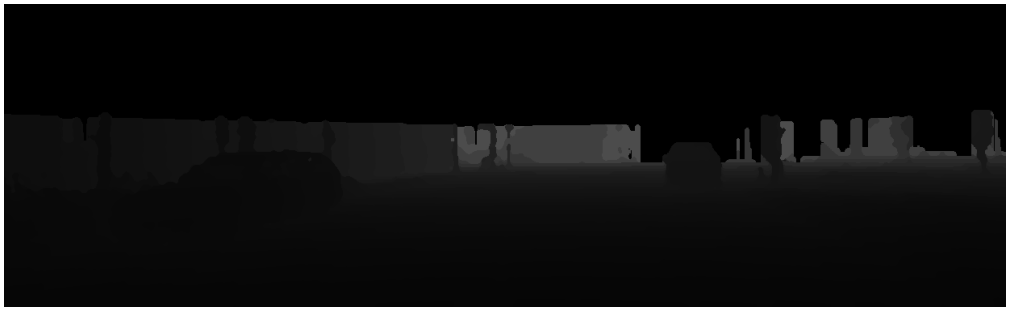

Image: 000014

Image: 000015

Note:

The depth maps were first trained on a training set of disparity images for which ground truth depth maps were available. The root mean square error (RMSE) for the calculated disparity maps and training images was calculated with all images having RMSE less than 2.

PART 2 – YOLO V3

For the second part of the assignment, we used the YOLOV3 detection algorithm to detect cars in the images. The YOLO weights were pretrained on the MS COCO data set. To implement the provided YOLO Code were required to select the threshold and bounding boxes for the given test images.

On the training images we selected the threshold as 0.5 and the confidence of 0.6. However, the confidence of 0.6 was too high for the test images as some of the cars were not highlighted (specially in Image 000011 where there are small cars in the center of the frame with low confidence).

For the test images the confidence threshold value was set 0.5 and the threshold value was set to 0.5 as well. This gave us the best results without any double bounding boxes. Additionally, all the cars in the images were identified as shown below.

Image: 000011

Image: 000012

Image: 000013

Image: 000014

Image: 000015

The values of the detection were saved in txt files to be read in the third section.

These are the following bounding box information that was saved based on read_label method that was provide in the Kitti Handler File.

Image	Detection Values Stored
000011	car 0 0 0 534 185 572 207 0 0 0 0 0 0 0 0 car 0 0 0 592 178 620 188 0 0 0 0 0 0 0 0
000012	car 0 0 0 482 186 544 231 0 0 0 0 0 0 0 0
000013	car 0 0 0 106 180 424 297 0 0 0 0 0 0 0 0 car 0 0 0 816 167 884 227 0 0 0 0 0 0 0 0
000014	car 0 0 0 501 176 540 207 0 0 0 0 0 0 0 0
000015	car 0 0 0 1037 172 1230 276 0 0 0 0 0 0 0 0 car 0 0 0 96 184 298 254 0 0 0 0 0 0 0 0

Note

some images have more than 1 car hence multiple output per images.

The 0 values are additional values that were not required in part 2 hence were set to 0.

PART 3- INSTANCE SEGMENTATION

Part 3 of the assignment required to do instance segmentation on the test images to mask all the cars that were detected using the YOLOV3 algorithm in part 2. The following general steps were used:

1. Loaded the estimated depth maps from part 1 of the assignment.
2. Loaded the yolo bounding boxes from part 2 of the assignment and x1, x2, y1, y2 values were extracted using labels_read method from Kitti handler file.
3. Created a numpy 2D array of '1's and multiplied all the values by 25 (to make a while image) of the shape of the depth map.

The training for the segmentation I experimented using different methods on the training images as ground truth was available and the method that gave me the best result was selected.

For segmentation we have 2 methods.

4.1 First method used is the mask library in NumPy (THIS WAS THE FINAL SELECTED METHOD)

a) Calculated the average of the depth values within the bounding boxes from part 2.
   i) Average was experimented using numpy.mean and numpy.nanmean (does not account for non zero values)
b) The mask was extracted using a distance threshold (See explanation for distance threshold experiment below)
c) Then the mask method was applied to extract locations where we had integer values.
d) The mask image from step c) then was converted in to black and white
e) The mask was applied to the blank white image we created above (step 2)

4.2 The second method was just using doing calculation based on matrices.

a) Calculated the average of the depth values within the bounding boxes from part 2.
   a. Average was experimented using numpy.mean and numpy.nanmean (does not account for non zero values)
b) The mask was extracted using a distance threshold (See explanation for distance threshold experiment below).** I used the where and logical_and method from NumPy library.
c) The mask was applied to the blank white image we created above (step 2)

5. The final segmentation images were saved as shown on the next page.

**

For distance threshold when calculating the mask there were 2 methods that I used.

1. Using 1 distance threshold value and applying on the bounding box depth map using
   a) average depth – distance threshold
   b) average depth + distance threshold
2. The second method we applied multiple distance threshold based on where the average depth lie in the range.
   a) We split the average depth in the following ranges based on training images.
      i) 0 to 8

ii) 8 to 10
iii) 10 to 15
iv) 15 to 20
v) 20 to 30
vi) 30 to 40
vii) 40 to 50
viii) 50 to 60
ix) 60+

b) Then a single threshold value was applied to the depth pixels to extract the mask.

3. For the third experiment we also split the minimum threshold and maximum threshold values which gave us the maximum flexibility.
   a) We split defined Minimum Threshold and Maximum based on the average depth (where it lied in the average range defined in 2. Above)

Using min/max threshold values for depth we were able to adjust our precision and recall during the training stage.

The training methods are shown in the Jupyter notebook as well.

Average Depth Range	Min	Max
0 - 8	2	3
8-10	3	2
10-15	3	3
15-20	10	15
20-30	8	8
30-40	8	9
40-50	5	7
50-60	10	15
60+	10	15

These are the following segmentation maps that were generated:

Image: 000011

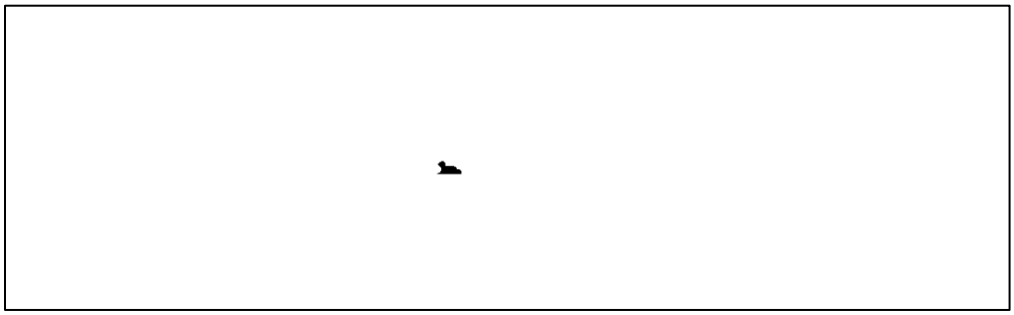

Image: 000012

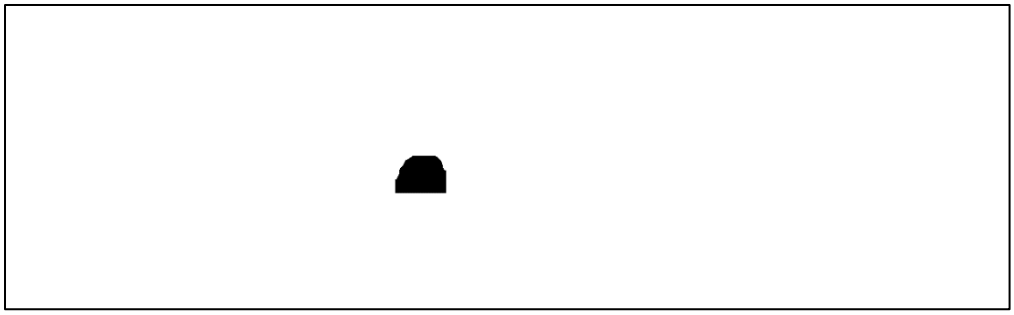

Image: 000013

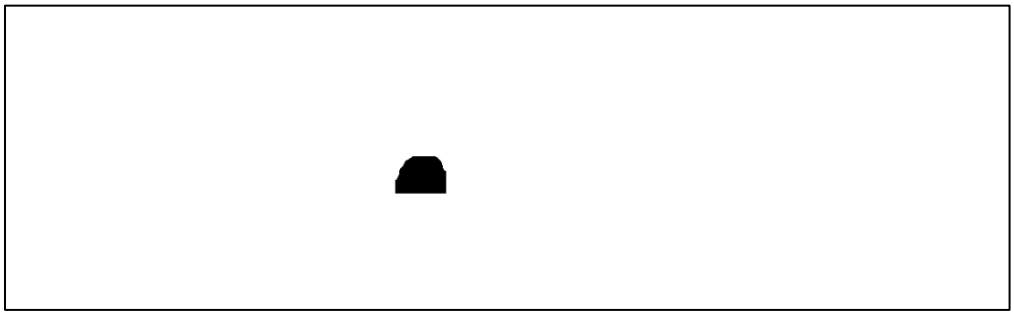

Image: 000014

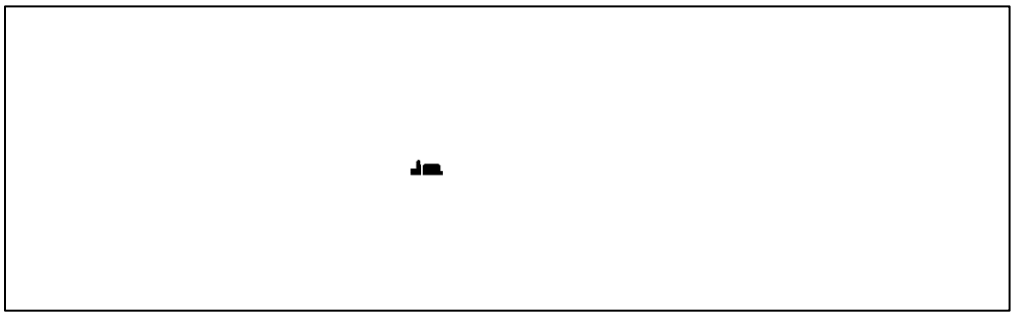

Image: 000015