Firstly, let's outline the development environment, as listed in table 1 below.

Development Environment		
Step	**Name**	**Details**
1	Ensure Python Version	3.6.10
2	Install Dependencies (pip install)	• opencv-python >= 4.4.0.44 • matplotlib >= 3.3.3 • numpy >= 1.19.2 • sklearn
3	Variables for Part 1 (Depth Estimation) **Execution**: `python part1_estimate_depth.py`	• TRAINING: True = use training data; False = use test data • VIS: True = For debugging, overlay depth estimation over left image; False = No visualizing
4	Variables for Part 2 (YOLOv3) **Execution**: `python part2_yolo.py`	• TRAINING: True = use training data; False = use test data
5	Variables for Part 3 (Instance Segmentation) **Execution**: `python python part3_segmentation.py`	• TRAINING: True = use training data; False = use test data • VIS: True = For debugging, visualize estimated segmentation mask; False = No visualizing

Table 1: Development Environment

1 Depth Estimation

For completeness, the predicted depth map are output for each image below. The equation to calculate depth from disparity is as follows:

$$depth = \frac{f * b}{disp}$$

where f b are the focal length and baseline of the stereo camera from its intrinsic data, respectively.

Figure 1: Depth Estimation for Test Image 000011

Figure 2: Depth Estimation for Test Image 000012

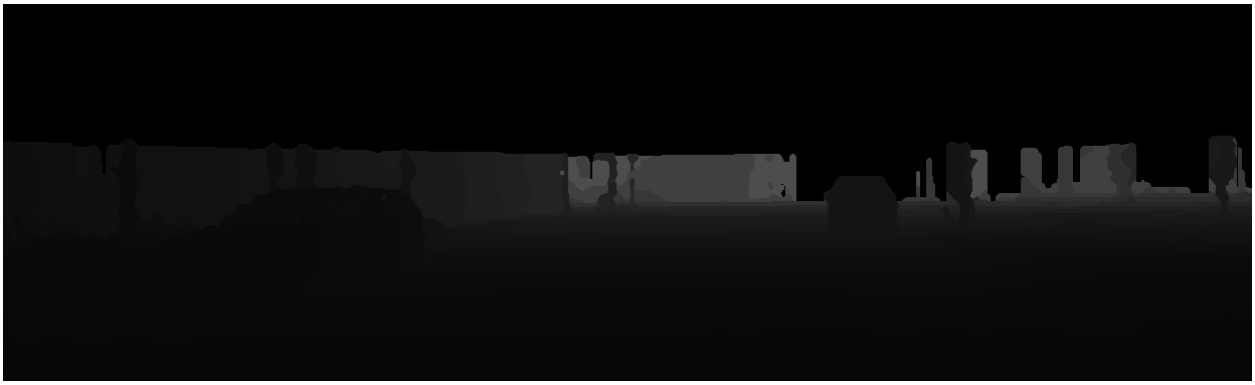

Figure 3: Depth Estimation for Test Image 000013

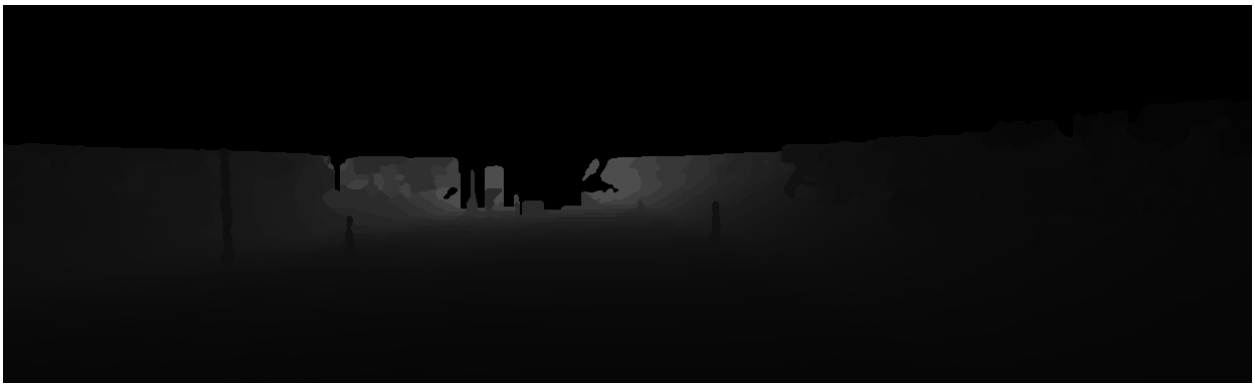

Figure 4: Depth Estimation for Test Image 000014

Figure 5: Depth Estimation for Test Image 000015

2 2D Bounding Box Estimation

In this section, the bounding boxes around estimated objects are included for the test image set below. Specifically, after iterating and empirically evaluating the results from the training data, a confidence threshold of 0.5 and NMS threshold of 0.4 were used.

Figure 6: 2D Object Detection Result for Test Image 000011

Figure 7: 2D Object Detection Result for Test Image 000012

Figure 8: 2D Object Detection Result for Test Image 000013

Figure 9: 2D Object Detection Result for Test Image 000014

Figure 10: 2D Object Detection Result for Test Image 000015

3 Instance Segmentation

In this section, the instance masks for the test image set are shown. Inspired from the assignment description, the instance masks were generated as mentioned in algorithm 1. Interestingly, since depth values larger than 80m are disregarded, it is quite possible that YOLOv3 recognizes a car beyond the depthmap's resolution (10cm - 80m), and as a result, any object labelled as a car, whose mean ROI is 0, their ROI will automatically be filled entirely as a mask for that car. The intuition behind this is that objects that are farther away, if recognized properly, will typically have a smaller bounding box than a larger one, so it is safe to trust YOLO's bounding box label at larger distances and fill the region entirely as a mask. Additionally, only the bottom have of the ROI obtained from part 2 was used to estimate the mean depth per object, and the intuition behind this is that the LIDAR data gets more sparse farther away from the ground due to less objects and more air, so the bottom half of the predicted ROI would give more stable and dense readings.

Algorithm 1: Instance Segmentation Approach

Result: Instance Segmentation Mask per Test Image
Given: estimated depth maps (part 1) and 2D BBox labels (part 2);
for *image in test set* **do**
  get its estimated depth map and labels from above;
  create clean output array of all 0's, call this *output*;
  for *label in all labels in this image named "car"* **do**
    get the bounding box from the label;
    apply bounding box to on image's depth map to get a region of interest (ROI);
    get the mean depth of the bottom half this depth ROI (ignore 0's);
    if *mean depth > 0* **then**
      $thresh$ = mean depth + $\epsilon(\epsilon = 1)$;
      apply OpenCV's binary threshold function on the ROI to get a mask (below thresh = 0 label, above = 255);
      apply this mask to appropriate ROI in *output*;
    else
      entire ROI is mask
    end
  end
  save mask;
end

For evaluating the performance on the training set, precision and recall were used against the ground truth mask.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where TP is true positive (a useful label in ground truth that matches with the estimated mask), FP is a false positive (predicted object mask with no associated ground truth), FN is a false negative (ground truth object with no predicted label). The precision and recall evaluation are shown in table 2.

Figure 11: Estimated Instance Mask for Test Image 000011

Figure 12: Estimated Instance Mask for Test Image 000012

Figure 13: Estimated Instance Mask for Test Image 000013

Figure 14: Estimated Instance Mask for Test Image 000014

Figure 15: Estimated Instance Mask for Test Image 000015

Precision and Recall Results on Training Data		
Training Sample	**Precision**	**Recall**
1	0.718	0.860
2	0.742	0.934
3	0.782	0.902
4	0.844	0.953
5	0.842	0.699
6	0.750	0.790
7	0.833	0.903
8	0.961	0.755
9	0.899	0.861
Average	0.816	0.860

Table 2: Precision and recall of estimated masks from training data