Midterm: 3D Object Detection

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Hi,
Good Job on your first submission. I can see your effort in finishing this. I like the way you implemented the visualization of BEV map, range, and intensity channel, and the way you compute the bev-map discretization. The function to let an open 3D window open until the right-arrow key is pressed is also well done.
I'd like to congratulate you. Please keep up the good work.

Tip

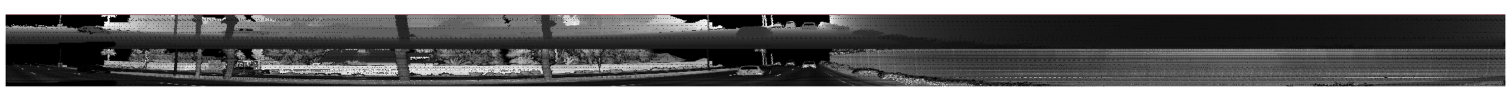
- In case you want to learn more about python best practices, this is a good place to do so.

Compute Lidar Point-Cloud from Range Image

✓

- Convert range image “range” channel to 8bit
- Convert range image “intensity” channel to 8bit
- Crop range image to +/- 90 deg. left and right of the forward-facing x-axis
- Stack cropped range and intensity image vertically and visualize the result using OpenCV

- The entire range of the data is mapped appropriately onto the 8bit channels of the range image and no data is lost. ✔



✓

- Visualize the point-cloud using the open3d module

Within a write-up file (Markdown or PDF):

- Find 10 examples of vehicles with varying degrees of visibility in the point-cloud
- Try to identify vehicle features that appear stable in most of the inspected examples and describe them

- Excellent work here. The point cloud has correctly been visualized with open3d.✔
- Vehicle features have been identified as well as the density of the 3d points.✔

Create Birds-Eye View from Lidar PCL

✓ Convert coordinates in x,y [m] into x,y [pixel] based on width and height of the bev map

Coordinates are converted correctly here. They are effectively shifted by half the image width to prevent negative coordinates

✓

- Assign lidar intensity values to the cells of the bird-eye view map
- Adjust the intensity in such a way that objects of interest (e.g. vehicles) are clearly visible

- The "intensity" channel of the BEV map with data from the point-cloud is filled correctly. The vehicles are not over-exposed in the bev-image with pixels at 255.
- The discretization is very well implemented. It is particularly fast. Well done.

✓

- Make use of the sorted and pruned point-cloud `lidar_pcl_top` from the previous task
- Normalize the height in each BEV map pixel by the difference between max. and min. height
- Fill the "height" channel of the BEV map with data from the point-cloud

Model-based Object Detection in BEV Image

✓ In addition to Complex YOLO, extract the code for output decoding and post-processing from the GitHub repo.

- Relevant code was extracted from GitHub and adapted where needed. .✔
- Objects are stored in the same way as the existing darknet model. ✔

✓

- Transform BEV coordinates in [pixels] into vehicle coordinates in [m]
- Convert model output to expected bounding box format [class-id, x, y, z, h, w, l, yaw]

The coordinate conversions are performed as the inverse operation to "Convert sensor coordinates to bev-map coordinates". Well done!

Performance Evaluation for Object Detection

✓

- For all pairings of ground-truth labels and detected objects, compute the degree of geometrical overlap
- The function `tools.compute_box_corners` returns the four corners of a bounding box which can be used with the Polygon structure of the Shapely toolbox
- Assign each detected object to a label only if the IOU exceeds a given threshold
- In case of multiple matches, keep the object/label pair with max. IOU
- Count all object/label-pairs and store them as “true positives”

Great Job using `tools.compute_box_corners` in finding the corner of the bounding boxes. The use of the polygon structure of the shapely toolbox to find the area of each bounding box is perfect.

Suggestion

Kindly check the following links to know more about IOU, and union in shapely:

- https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
- https://shapely.readthedocs.io/en/stable/manual.html

✓ Compute the number of false-negatives and false-positives based on the results from IOU and the number of ground-truth labels

Good work using the number of valid labels and not the number of all labels to compute the number of all positives.

✓

- Compute “precision” over all evaluated frames using true-positives and false-positives
- Compute “recall” over all evaluated frames using true-positives and false-negatives

Setting the flag use_labels_as_objects to true to check whether both precision and recall result to 1.0. ✔

DOWNLOAD PROJECT

RETURN TO PATH