Perception

Team 16

December 2017

With contributions from: Brad Saund, Sagar Israni, Alex Rochaix, Justin Diep, Joe Abrash, Matthew Romano, Shreyas Sudhakar, Zaid Ashai, Mattison Rose.

1 Overview

The purpose of this algorithm is to identify how many vehicles are in a given image and its corresponding lidar point cloud. This is accomplished through a two phase pipeline. Phase 1 extracts clusters of points from the Lidar data. Phase 2 uses a retrained image classifier on the sub-images corresponding to each clusters of interest.

2 Phase I - Lidar Clustering

First, the image and point clouds are extracted from the dataset, and additional lidar points are artificially added to densify the point cloud and fill in far away gaps. The data is then filtered to identify areas of interest. Ground and uninteresting points, (those unlikely to be lidar car data points), are removed. Then the point cloud is segmented into clusters using Euclidean segmentation. (Note: The densification described earlier prevents false over-segmentation of clusters due to Lidar resolution). Clusters that are too big or small to be cars and clusters that have too severe an aspect ratio to be a car are removed, leaving clusters of points that are potentially, but not necessarily, cars.

3 Phase II - Image Classification

Sub-images are generated for each cluster of interest by projecting the points in the cluster to the image frame, and cropping the image appropriately. A Tensorflow Neural Net then classifies these images of interest into "cars" or "not cars" and counts the number of cars in the whole image. The publicly available Inception v3 tensorflow convolutional neural network (pre-trained on imagenet) is utilized but with a re-trained final layer using images extracted and labeled from the provided training data using Phase I. The output is then plotted.

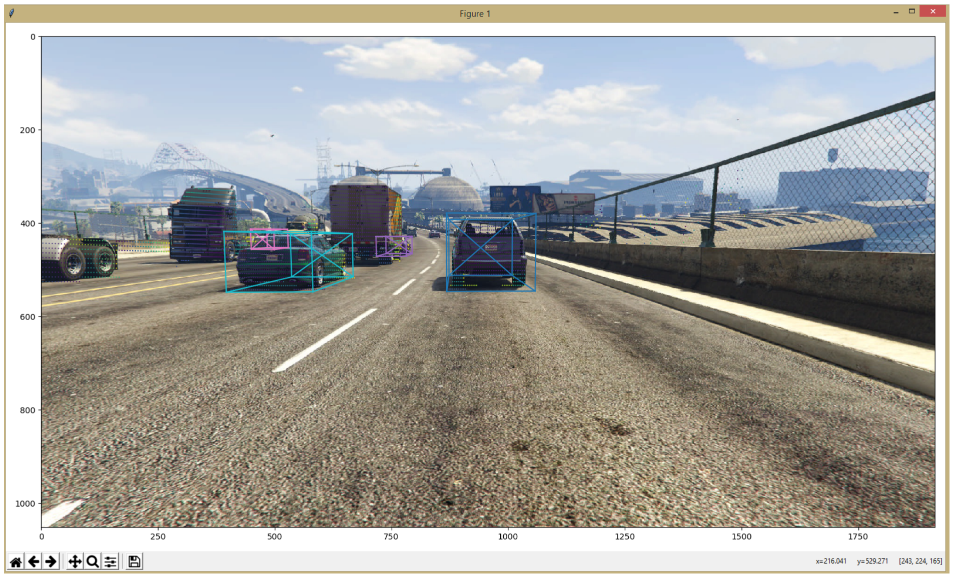
Figure 1: Training set image, with filtered lidar data added.

4 Appendix 1: Algorithm Pseudocode

Initial setup:

1. Extract the classes of interest from the tensorflow neural network

2. Open training dataset.and extract lidar data points, camera projection matrix, and bounding boxes.

3. Additional lidar points are added to fill in far away gaps with:

   ```
   parse_lidar.densify_lidar
   ```

   - This is used later as we will cluster by distance points in the same radial area to identify individual objects. Raw lidar data points that are far away from the sensor will be far from any other measurements.
   - This is useful because distant objects will be detected by raw lidar data with low fidelity.

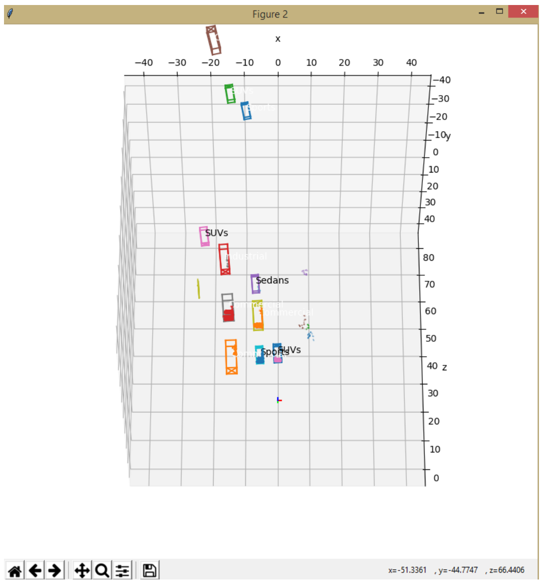

Figure 2: Plotting lidar data with clusters of interest and cars identified.

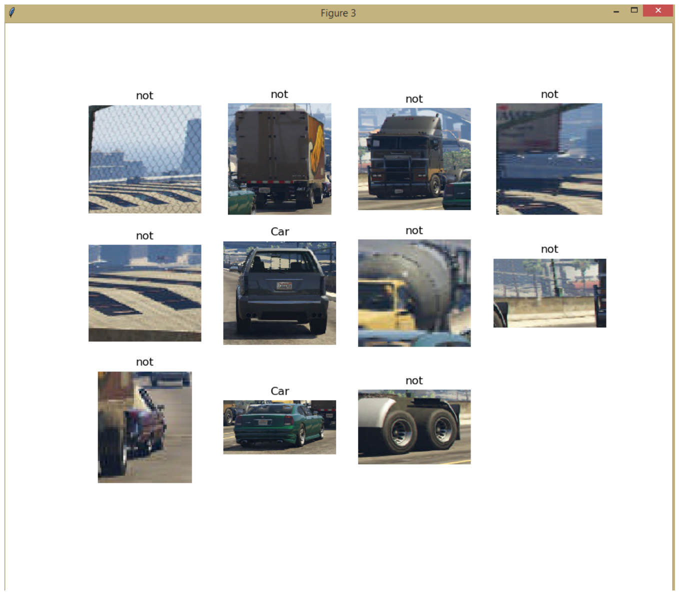

Figure 3: ‘Cars’ and ‘not’ cars identified.

Algorithm 1 Detect Cars

```
for all images do
    1. Filter lidar data to identify images of interest with:
            parse_lidar.py

    2. Remove ground points and uninteresting data e.g. points likely to not be lidar car data points
    with:
            mask_out_horizontal , mask_far_points , lidar_mask

    3. We then segment the point cloud into clusters where each cluster is at least a set minimum
    distance apart to differentiate objects with:
             parse_lidar.euclid_segmentation.

    4. Remove clusters that are too big or small to be cars with:
            parse_lidar.car_clusters

    This leaves clusters of points that are potentially cars with:
            get_points_of_interest.

    5. Sub-images are generated that correspond to the clusters of interest with:
            parse_lidar.extract_image , parse_lidar.extract_images

            parse_lidar.get_imgs_and_clusters

    6. Images of interest that have too severe an aspect ratio to be a car are removed with:
            extract_images

    7. Use tensorflow to classify these images of interest, counting the number of cars
end for=0
```

5 Appendix 2: Setup to run the code

Code available on: https://github.com/bsaund/Rob599Perception
Use the code on the master branch. Follow the instructions on the github README.md (copied below)

1. Setup on linux
   - cd
   - virtualenv -p python3 tf3
   - source /tf3/bin/activate
   - sudo apt install python3-tk
   - pip install tensorflow matplotlib pillow scikit-learn imageio networkx
2. Download the training data and place in a folder next to (out of) the repo.
3. Run:

   ```
   python ./count_test
   ```

   to start computing labels for the test data. This will write to outfile.txt.
4. To view this file in real time:

   ```
   tail -f outfile.txt
   ```

5. To play around with the training data, run:

   ```
   python ./tmp
   ```

6. To retrain the tensorflow network:
   - Look at Tensorflow for poets

     ```
     https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0
     ```

   - Generate training images (will take several hours) with:

     ```
     python generate_labeled_training_data.py
     ```

   - Metafile
   - Put the data in correct format with:

     ```
     python organize_labeled_data.py
     ```