Lane Quality Assessment for Autonomous Driving

Binbin Li[*], Dezhen Song[*], Haifeng Li[†], Adam Pike[‡], and Paul Carlson[§]

November 15, 2018

We describe a C++ framework to quantify the lane quality to assist autonomous driving in urban scenarios. We propose different metrics and and algorithms by detecting and tracking immediate left and right lane boundaries with respect to the vehicle due to their importance in guiding the vehicle. We do not identify multiple parallel lanes simultaneously because the onboard sensors have perspective limitations.

The vehicle is equipped with one front view camera, one light detection and ranging (LIDAR) system, and GPS/IMU units, which are common sensory configurations for AVs. We have the following assumptions.

- The camera is pre-calibrated with known intrinsic parameters and nonlinear distortion removed.
- Sensor readings are already synchronized.
- The coordinate system transformations between any two sensors are known by calibration.

It is worth noting that our problem is not the same as the well-known lane marking detection problem. It is not in our best interest to develop/apply the most sensitive and accurate lane marking detection algorithm because we want to ensure that our roads are safe for less capable vehicles. *Here we measure roads instead of vehicles*. We focus on evaluating lane qualities instead of abilities to detect them. It needs to be able to output different types of quality issues instead of just reporting "no lane" detected.

We aim to use the framework:

- To provide an easy-to-extend and easy-to-use general library for lane quality assessment, lane marking detection or obstacle avoidance that can be easily applied to different problems in autonomous driving,
- To provide people who want to an easy-to-read implementation that focuses on the relevant details of the problem specification.
- to help user to browse/extend the code.

The current framework is able to provide,

- Lane marking pixel's coordinates in the image;
- 3D lane marking points;
- 3D B-spline lane boundary curve models.
- Lane quality metrics.

[*]B. Li and D. Song are with Department of Computer Science and Engineering, Texas A&M University, College Station, TX, 77843, US. (email: binbinli@tamu.edu, dzsong@cse.tamu.edu)

[†]H. Li is with Computer Science Department, Civil Aviation University of China, Tianjin, 300300, China, and also with Fujian Provincial Key Laboratory of Information Processing and Intelligent Control, Minjiang University, Fuzhou, 350108, China. (email:hfli@cauc.edu.cn)

[‡]A. Pike is with Texas A&M Transportation Institute, College Station, TX, 77843, US. (email: A-Pike@tti.tamu.edu)

[§]P. Carlson is with Road Infrastructure, Inc., College Station, TX, 77843, US. (email: PCarlson@roadinfrastructureinc.com)

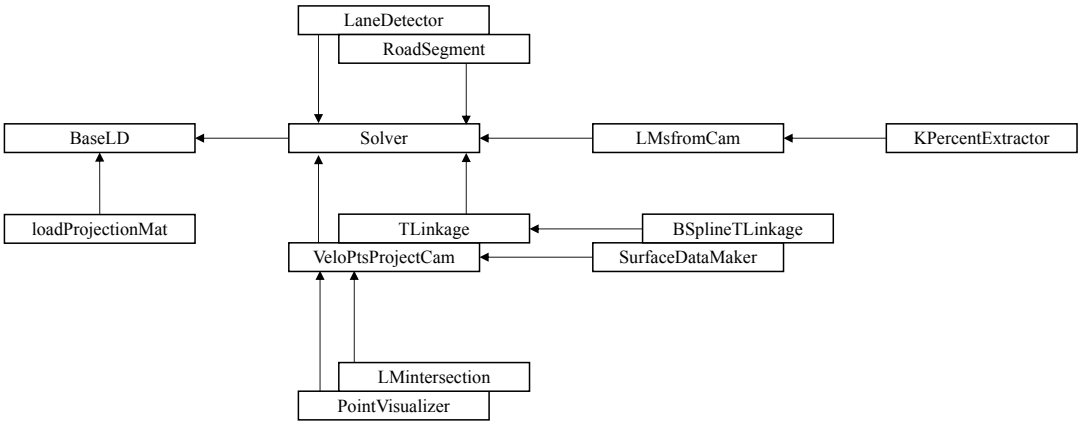

Figure 1: Class diagram for the proposed framework. The solid arrow indicates "is-a" relationship.

The code is released under the GPLv3 license. It has been tested on Ubuntu 16.04LTS and should work on 16.04 or newer versions, and verified using KITTI datasets [1]. This code is based on the paper by Binbin Li, Dezhen Song, Haifeng Li, Adam Pike, and Paul Carlson, "Lane Marking Quality Assessment for Autonomous Driving", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, October, 1-5, 2018 [2]. For detailed installation, plase look into the README.md in the source code.

1 Library Overview

We refer to the class diagram in Fig.1 to visualize the interplay of the framework. It mainly contains the following classes,

- `loadProjectionMat` load KITTI's `calib_velo_to_cam.txt` and `calib_cam_to_cam.txt` to get the projection matrix $\mathbf{P}$ from LIDAR to camera. The user/reader is required to change the `Parameter.xml` to locate those files.
- `RoadSegment` obtain the road surface image mask [3] from color images through ICNet [3]. It loads the model file `icnet_cityscapes_train_30k.caffemodel` from the folder `/model` and the configuration file `icnet_cityscapes.prototxt` in the folder`/prototxt`. Change the class constructor in class `RoadSegment` if you want to deploy different models and configuration files.
- `LMsfromCam` detect lane marking pixels and output image mask for lane markings from images. The user/reader is required to change the `Parameter.xml` in the source code to locate the images.
- `VeloPtsProjectCam` project 3D LIDAR point cloud into the image using $\mathbf{P}$ from `loadProjectionMat`.
- `LMintersection` intersect lane marking points from image and the corresponding LIDAR scan with the same timestamps at LIDAR system coordinate in order to determine the boundary models.
- `SurfaceDataMaker` display the projected LIDAR points belonged to the road surface on image by using the image mask from class `RoadSegment`.
- `KPercentExtractor` extract lane markings from image and refine the results through vanishing point detection.
- `TLinkage` a multi-model fitting approach.
- `BSplineTLinkage` B-spline lane boundary curve model.
- `LaneDetector` detect lane markings from the image and LIDAR scan, obtain the lane boundary models, and compute the lane quality metrics.

For sample source data and experimental results, please look into the `data` folder. This documentation is by no means complete, but it is intended to help the user browsing the C++ code.

```
<?xml version="1.0" encoding="UTF-8"?>
  <Document>
    <Solvers>
    ......
      <LaneDetector
      ......
        lassAsses="false"
      ......
      />
    ......
    <!--change the solver to execute the functional block-->
    <Main solver="LaneDetector" debug="true"/>
</Document>
```

Listing 1: Change the configuration file to select the functionality.

1.1 Configuration File

The framework requires two command line arguments (see `main.cpp`), and the second argument is the configuration file, i.e., `Parameters.xml`, which contains the environment settings for the whole framework. Through the configuration file, the user/reader can choose the folders to read source data, set threshold values for each classes, and output experimental results. Several important parameters are provided below,

- `dataRoot` the path to the folder that contains a sequence of RGB images collected from the front-view camera.
- `dataFile` the path to the file that contains a set of image's name that the user/reader is interested to get the quality metrics or the lane models.
- `veloRoot` the path to the folder that contains a sequence of binary files, which contains a set of LIDAR points.
- `segImgPrefix` the name prefix for segmented road surface image mask for `RoadSegment` in Fig.1. The image mask, a binary image, indicates the road surface pixel coordinates in the image.
- `overlayedRoot` the directory to store the color images by overlaying the image mask with the original image for visualization.
- `refinedRoot` the directory to store the lane marking's image mask.
- `refinedImgPrefix` the name prefix for the lane marking's image mask.
- `minSamples` the minimum number of 3D LIDAR points for the B-spline lane boundary models.
- `ratiosFile` the path to the file to store the lane quality metrics.
- `modelFile` the path to the file to store the B-spline lane boundary curve mode.

The framework loads the configuration file, and parses the parameter setting into different classes for initialization (see Listing 2).

1.2 Functionality Selection

Through pugixml library [4], we provide a flexible easy-to-use structure to execute different functionalities by changing the "solver"'s name in the configuration file (see Listing 1). Replace the "solver"'s name to accomplish the target task without repeatably executing all the other functionalities. It greatly decreases the debugging time and increase the codding efficiency. The only requirement is that the user/reader needs to store all the intermediate results from all the previous steps. For example, the `RoadSegment` provides road surface image mask for `LMsfromCam`, which is required to be saved as input for the class `LMsfromCam`. The `main.cpp` parses the "solver"'s name, select the classes listed in Listing 2,

```
int main(int argc, char *argv[]) {
  if (argc != 2)
  throw runtime_error("ERROR: xml file is expected!");

  pugi::xml_document xml;
  pugi::xml_parse_result status = xml.load_file(argv[1]);
  if (!status)
  throw runtime_error(string("xml error") + status.description());

  string solver = xml.document_element().child("Main").attribute("solver").as_string();
  std::unique_ptr<LD::Solver> solverPtr = nullptr;

  if (boost::iequals(solver, "RoadSegment"))
    solverPtr = std::make_unique<RoadSegment>(argv[1]);
  else if (boost::iequals(solver, "KPercentExtractor"))
    solverPtr = std::make_unique<KPercentExtractor>(argv[1]);
  else if (boost::iequals(solver, "LMsintersection"))
    solverPtr = std::make_unique<LMsintersection>(argv[1]);
  else if (boost::iequals(solver, "BSplineTLinkage"))
    solverPtr = std::make_unique<BSplineTLinkage>(argv[1]);
  else if (boost::iequals(solver, "Line3DTLinkage"))
    solverPtr = std::make_unique<Line3DTLinkage>(argv[1]);
  else if (boost::iequals(solver, "LaneDetector"))
    solverPtr = std::make_unique<LaneDetector>(argv[1]);
  else
    throw runtime_error("No such solver implemented: " + solver);

  solverPtr->Run();

  return 0;
}
```

Listing 2: Parse Parameter.xml to execute different functionality in main.cpp.

and execute the corresponding one. We suggest to follow the steps, `RoadSegment -> KPercentExtractor ->LMsintersection ->BSplineTLinkage`, to get the intermediate results such as road mask, lane marking points in the LIDAR scan, and lane boundary model. The class `LaneDetector` will execute all the above steps. For complete analysis, replace the "solver"'s name with "`LaneDetector`" like Listing 1 and perform road surface segmentation to lane quality metric computation.

The current framework provides the capabilities to switch between lane quality assessment mode and lane detection mode. If `lassAsses="false"` in Listing 1, the framework will stop computing the quality metrics. The user/reader decides the way of exploiting the framework and has the freedom to choose the functionality according to different application scenarios.

We also have a user-defined debug mode (see `m_debug` in class `BaseLD`) to provide additional runtime information of the framework. Change `debug ="true"` in Listing 1 to `debug ="false"` to stop the user-defined debug mode.

1.3 Error Capture

If the current lane that the autonomous vehicle is driving on does not have any lane markings, the system outputs `WARNING: Could not find enough samples!` in the `ratio.txt` file. Other possible runtime errors are also captured as well. Please look into class `LaneDetector` for detailed implementation.

2 Tools

2.1 Compatibility with the Velodyne LIDAR System

For the `pcap` file or live stream from Velodyne 16/32E/64E, please refer to the folder `3rdParty/pcap2kitti` to convert to the binary files. For `pcap` file, run `pcap2kitti filename.pcap` and obtain binary files frame by frame. For live stream output from the Velodyne, change the IP address and port number in `pcap2kitti.cpp`. Noted we provide standalone package to convert Velodyne output to KITTI's

binary format. It is not integrated with the lane quality assessment framework for the purpose of convenience of the usage. If complete integration is required, change the function `ReadVeloData` in class `VeloPtsProjectCam`.

2.2 Vanishing Point Detection

We employ line segment detection [5] for vanishing point detection to refine lane marking detection results from the images. Detailed usage is in the class `KPercentExtractor`.

2.3 Lane Boundary Model Compatibility

We use T-linkage [6], a multi-model fitting approach, to obtain the lane boundary models. We apply cubic B-spline curve models into the T-linkage to get left and right lane boundaries. We also provide 3D line lane models in folder `3rdParty/TLinkage`. To use the line models, simply change the "solver" name to `Line3DTLinkage` in the configuration file `Parameters.xml` (see Listing 1).

3 Contributors

Dezhen Song	Project Director, Concept Design
Binbin Li	System Design, Primary Software Developer
Aaron Kingery	Software Developer, Tester, and Documentation
Aaron Angert	Software Developer, Tester, and Documentation
Ankit Ramchandani	Software Developer, Tester, and Documentation

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[2] Binbin Li, Dezhen Song, Haifeng Li, Adam Pike, and Paul Carlson. Lane marking quality assessment for autonomous driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October, 1-5, 2018.

[3] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. *arXiv preprint arXiv:1704.08545*, 2017.

[4] Arseny Kapoulkine. pugixml: Light-weight, simple and fast xml parser for c++ with xpath support, 2012.

[5] Rafael Grompone Von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: a line segment detector. *Image Processing On Line*, 2:35–55, 2012.

[6] Luca Magri and Andrea Fusiello. T-linkage: A continuous relaxation of j-linkage for multi-model fitting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3954–3961, 2014.